Copy No. _____

Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

C++ classes for representing airfoils

David Hally

Defence R&D Canada – Atlantic

Technical Memorandum
DRDC Atlantic TM 2009-053
January 2010

Canada

C++ classes for representing airfoils

David Hally

Defence R&D Canada – Atlantic

Technical Memorandum

DRDC Atlantic TM 2009-053

January 2010

Principal Author

Original signed by David Hally

David Hally

Approved by

Original signed by D. Hopkin

D. Hopkin
Head/Maritime Asset Protection

Approved for release by

Original signed by Ron Kuwahara for

C. Hyatt
Head/Document Review Panel

Abstract

A library of C++ classes for representing the geometry of airfoils is described. The classes are based on the CurveLib library for representing differentiable curves. Airfoils that have been represented explicitly include Joukowski airfoils, the NACA 4-digit, 5-digit, 16-series and 6-series airfoils, as well as the DTMB modification of the NACA 66 airfoils commonly used for marine propellers. Generic airfoils can be defined from offsets on the airfoil surface or from offsets defining mean line and thickness curves. Several methods for closing the trailing edges of airfoils with a trailing edge gap are also implemented.

Résumé

Le présent document décrit une bibliothèque de classes C++ permettant de représenter la géométrie des surfaces aérodynamiques. Les classes de la bibliothèque CurveLib permettent de représenter des courbes sous forme de fonctions différentiables. Les surfaces aérodynamiques qui ont été représentées comprennent les profils de Joukowski, les profils NACA à 4 et à 5 chiffres, les profils de la série 16 et de la série 6, ainsi que la modification DTMB des profils NACA 66, utilisés couramment pour les hélices de navire. Les profils génériques peuvent être définis à l'aide de décalages par rapport à la surface aérodynamique ou de décalages définissant la corde moyenne et les courbes d'épaisseur. Plusieurs méthodes visant à terminer le bord de fuite des profils par un écart sont également mises en œuvre.

This page intentionally left blank.

Executive summary

C++ classes for representing airfoils

David Hally; DRDC Atlantic TM 2009-053; Defence R&D Canada – Atlantic; January 2010.

Background: The flow around marine propellers affects their performance and the noise that they produce. Defence R&D Canada – Atlantic uses Computational Fluid Dynamics (CFD) to calculate these flows so that the performance and noise of propellers can be evaluated and improved. Before the flow can be calculated, the geometry of the propeller must be represented in a fashion that can be used by the CFD applications. The blade of a propeller is usually constructed from a series of airfoil sections. The current document describes a library of C++ classes which can be used to represent these airfoils so that they can be used as building blocks for the generation of propeller geometry.

Principal results: A library of C++ classes for representing airfoils has been written. The classes are based on the CurveLib library for representing differentiable curves. Airfoils that have been represented explicitly include Joukowski airfoils, the NACA 4-digit, 5-digit, 16-series and 6-series airfoils, as well as the DTMB modification of the NACA 66 airfoils commonly used for marine propellers. Generic airfoils can be defined from offsets on the airfoil surface or from offsets defining mean line and thickness curves. Methods are implemented for closing the trailing edges of airfoils with a trailing edge gap.

Significance: The library of C++ classes provides a useful tool for representing the geometry of airfoils for use in CFD programs. They are used as elements for the representation of the more complex geometry of marine propellers.

Sommaire

C++ classes for representing airfoils

David Hally ; DRDC Atlantic TM 2009-053 ; R & D pour la défense Canada – Atlantique ; janvier 2010.

Introduction : L'écoulement autour des hélices de navire a une incidence sur leur rendement et sur le bruit qu'elles produisent. R & D pour la défense Canada – Atlantique utilise la dynamique des fluides numérique (CFD) pour calculer ces écoulements, de sorte que le rendement et le bruit des hélices peuvent être évalués et améliorés. Avant de calculer l'écoulement, il faut représenter la géométrie des hélices de manière à ce qu'elle puisse être utilisée dans les applications CFD. Les pales des hélices sont habituellement composées de plusieurs éléments qui présentent chacun une surface aérodynamique distincte. Le présent document décrit une bibliothèque de classes C++ permettant de représenter la géométrie de ces surfaces aérodynamiques, de sorte qu'elles peuvent être utilisées comme éléments structuraux pour générer la géométrie des hélices.

Résultats : Une bibliothèque de classes C++ permettant de représenter la géométrie des surfaces aérodynamiques a été créée. Les classes de la bibliothèque CurveLib permettent de représenter des courbes sous forme de fonctions différentiables. Les surfaces aérodynamiques qui ont été représentées comprennent les profils de Joukowski, les profils NACA à 4 et à 5 chiffres, les profils de la série 16 et de la série 6, ainsi que la modification DTMB des profils NACA 66, utilisés couramment pour les hélices de navire. Les profils génériques peuvent être définis à l'aide de décalages par rapport à la surface aérodynamique ou de décalages définissant la corde moyenne et les courbes d'épaisseur. Plusieurs méthodes visant à terminer le bord de fuite des profils par un écart sont également mises en ?œuvre.

Portée : La bibliothèque de classes C++ est un outil pratique pour représenter la géométrie des surfaces aérodynamiques à utiliser dans les programmes de CFD. Elles sont utilisées comme éléments pour représenter la géométrie plus complexe des hélices de navire.

Table of contents

List of figures

1 Introduction

The flow past airfoils is important in many aeronautical and marine applications. This document describes a library of C++ classes, hereafter called the Airfoil library, which can be used to represent the geometry of airfoils so that the flow around them can be calculated using Computational Fluid Dynamics (CFD) programs. The classes are based on the CurveLib library [1] for representing differentiable curves.

Several families of airfoils are represented explicitly in the Airfoil library: Joukowski airfoils (Section 5.2), National Advisory Committee for Aeronautics (NACA) 4-digit, 5-digit, 16-series and 6-series airfoils [2] (Section 6.1.3), and the David Taylor Model Basin (DTMB) modification of the NACA 66 airfoil commonly used in marine propellers (Section 6.1.4). Generic airfoils can be generated from offsets on the airfoil surface (Section 5.1), or from mean lines and thickness curves (Section 6.1). The mean lines and thickness curves can themselves be defined from explicit representations (Sections 6.1.1.2–6.1.1.5 and 6.1.2.1 or from offsets (Sections 6.1.1.6 and 6.1.2.2).

Methods for closing the trailing edges of airfoils with trailing edge gaps are also implemented (Section 6.1.2 and Annex B).

Classes are also provided for defining airfoils from data in a file in OFFSRF format [6] (Section 7) and for specifying an airfoil from the program command line in a Unix/Linux shell or DOS command window (Section 8).

2 Airfoil geometry

The Airfoil library allows the surface of an airfoil to be parameterized in two ways:

- Using ξ, a parameter which is 0 at the trailing edge on the pressure side, $\frac{1}{2}$ at the leading edge and 1 at the trailing edge on the suction side. The surface of the airfoil using this parameterization is $\mathbf{x}(\xi)$; this will be called the *complete airfoil curve*. Normally ξ is approximately equal to the fractional arclength along the airfoil surface (since the leading edge must be at $\xi = \frac{1}{2}$, it cannot be exactly equal to the fractional arclength unless the pressure and suction sides of the airfoil have the same length).

- Using ζ, the fractional chord length along the straight line joining the leading edge ($\zeta = 0$) to the trailing edge ($\zeta = 1$). This parameterization requires that the airfoil surface be split into two different curves: $\mathbf{p}(\zeta)$ for the pressure side and $\mathbf{s}(\zeta)$ for the suction side. These curves will normally have undefined derivatives at the leading edge.

These parameters are illustrated in Figure 1.

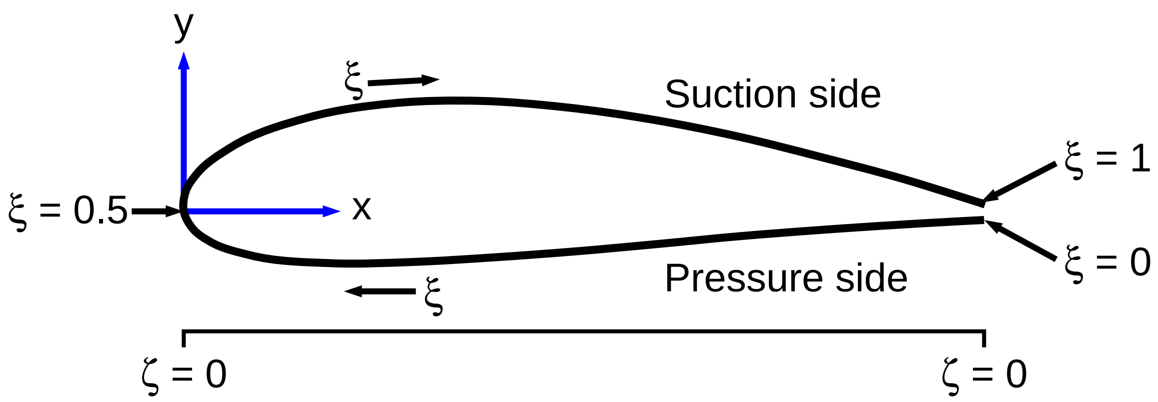

Figure 1: *The airfoil coordinate system and parameters. The origin of the coordinates is shown as if the airfoil is canonical. This airfoil has an open trailing edge.*

The location of the leading edge, $\mathbf{x}_{le}$, cannot be determined simply from the geometry of the airfoil. Instead it is *defined* to be $\mathbf{x}(\frac{1}{2})$, $\mathbf{p}(0)$ or $\mathbf{s}(0)$ (for a well-defined airfoil these points should all be the same).

The trailing edge, $\mathbf{x}_{te}$, is $\frac{1}{2}\big(\mathbf{x}(0) + \mathbf{x}(1)\big)$ or $\frac{1}{2}\big(\mathbf{p}(1) + \mathbf{s}(1)\big)$ (for a well-defined airfoil these points should be the same).

The chord line is the straight line connecting the leading edge and the trailing edge. The chord length is the length of the chord line: i.e. the distance between the leading and trailing edges.

The mean line is the line equidistant from the pressure side and suction side curves. It is also difficult to define from the geometry alone. Airfoil classes are free to define it to suit their purposes provided that it lies within the airfoil and includes the leading and trailing edges. The default definition of the mean line is

$$\mathbf{m}(\zeta) = \tfrac{1}{2}\big(\mathbf{p}(\zeta) + \mathbf{s}(\zeta)\big). \tag{1}$$

However, note that with this definition the derivative at the leading edge is not well-defined as the derivatives of the pressure and suction side curves are not well-defined there.

It is possible for an airfoil to be open at the trailing edge: i.e. $\mathbf{x}(0) \neq \mathbf{x}(1)$ and $\mathbf{p}(1) \neq \mathbf{s}(1)$. If it is closed, it may be blunt (i.e. the unit normal at the pressure side trailing edge is the same as the unit normal at the suction side trailing edge) or sharp (the unit normals at the trailing edge differ). The airfoil classes provide member functions for determining whether the trailing edge is closed or open, blunt or sharp, and for closing trailing edges in various ways. If the trailing edge is closed and blunt, the curves $\mathbf{p}(\zeta)$ and $\mathbf{s}(\zeta)$ will have undefined derivatives when $\zeta = 1$.

It is normal to define airfoils so that the leading and trailing edges lie at (0,0) and (1,0) respectively; the airfoil is then called canonical. The airfoil classes do not require an airfoil to be canonical but include a function that will scale and rotate any airfoil so that it is.

3 Subsidiary classes

There are a number of classes used by the Airfoil library that have previously been described in other reports. They are described briefly here.

`VecMtx::VecN<N,F>`
: A vector of `N` elements of type `F` where `F` is usually a `float` or a `double`. `VecMtx::VecN<N,F>` implements a full range of arithmetic operators. See Reference 1, Annex B for a complete description of the class.

`CurveLib::Curve<N,V,F>`
: A differentiable curve having `N` arguments of type `F` and which returns a value of type `V`. This class and its many derived classes are described in Reference 1.

3.1 Exceptions

All exceptions thrown by Airfoil classes and functions are derived from the base class `Error` in the global namespace. An `Error` contains a message which can be retrieved, appended to, or prepended to. The prototypes of the `Error` member functions are listed in Reference 1, Annex F.

It is wise, when using the Airfoil library classes, to enclose the body of the code in a `try` block which catches an `Error`. For example:

```
try {
  ... // Code which uses Airfoil library classes
}
catch (Error &e) {
  // Write the error message
  std::cerr << e.get_msg() << '\n';
}
```

Another important exception is `ProgError`, a specialization of `Error`. It is thrown when an exception occurs that can clearly be recognized as a programming error rather than a error by the user. The occurrence of a `ProgError` is an indication that the program is faulty. The prototypes of the `ProgError` member functions are listed in Reference 1, Annex F.1.

4 The base airfoil class

All classes in the Airfoil library are encapsulated in the namespace `Afoil`.

All airfoils are derived from the base class `Airfoil<F>` which is itself derived from `Curve<1U,VecMtx::VecN<2U,F>,F>`: that is, it is a one-parameter curve which returns a 2-vector representing a point on the airfoil surface. This curve is a representation of $\mathbf{x}(\xi)$: i.e. each parameter is interpreted as a value of ξ. Each value of ξ has type `F` and each component of the returned point also has type `F`. Usually `F` is `float` or `double`.

`Airfoil<F>` provides the following alias for the type of the returned point:

```
typedef VecMtx::VecN<2U,F> Point
```

The airfoil curve represented by an instance of `Airfoil<F>` may be evaluated using the following two member functions:

`Point operator()(F xi) const`
: Returns the value of $\mathbf{x}(\xi)$ at $\xi =$ `xi`.

`Point operator()(F xi, unsigned d) const`
: Returns the value of the differentiated airfoil curve at `xi`. The number of derivatives to be taken is specified by `d`. If `d` is zero, then this function is equivalent to `operator()(F xi) const`.

For example, suppose that we defined a NACA 0012 airfoil by

```
using namespace Afoil;
NACAAirfoil<double> naca_afoil("0012");
```

To evaluate the point on the airfoil at $\xi = 0.25$ use:

```
Airfoil<double>::Point p = naca_airfoil(0.25);
```

To evaluate a tangent to the airfoil surface at the same value of ξ we could use:

```
Airfoil<double>::Point tangent = naca_airfoil(0.25,1);
```

All curves derived from `Airfoil<F>` in the Airfoil library have a default constructor: i.e. a constructor having no arguments. The airfoil remains undefined if the default constructor is used. It can later be defined using specialized member functions in the derived class or by assignment to another airfoil.

Since an airfoil may remain undefined if a default constructor is used, `Airfoil<F>` provides the following member function for determining whether the curve is defined or not (it is actually inherited from `Curve<N,V,F>`).

`bool is_defined() const`
: Returns `true` if the airfoil has been defined; `false` if it has not.

An attempt to evaluate an undefined airfoil will cause an `Error` exception to be thrown (see Section 3.1).

An important property of classes derived from `Airfoil<F>` is that they are polymorphic, even though the class `Airfoil<F>` has no virtual functions. For example, suppose a `NACAAirfoil<double>` is assigned to an `Airfoil<double>`:

```
using namespace Afoil;
NACAAirfoil<double> naca_afoil("0012");
Airfoil<double> afoil;
afoil = naca_afoil;
```

When evaluated with the same arguments, `afoil` will return the same value as `naca_afoil`.

`Airfoil<F>` has the following members in addition to the default constructor, copy constructor, destructor and assignment operator.

`typedef CurveLib::Curve<1U,Point,F> CurveType`
: The types of the curves defining the surface of the airfoil. These curves take a scalar parameter (either ξ or ζ) for an argument and return a point in two-dimensional space.

`typedef CurveLib::Curve<1U,F,F> ParamCurveType`
: The type of the curves relating the curve parameters ξ and ζ. These curves take a scalar value for an argument and return a scalar value.

`CurveType suction_side() const`
: Returns a curve defining the suction side of the airfoil, $\mathbf{s}(\zeta)$. The curve is parameterized by the fractional chord length, ζ.

`CurveType pressure_side() const`
: Returns a curve defining the pressure side of the airfoil, $\mathbf{p}(\zeta)$. The curve is parameterized by the fractional chord length, ζ.

`CurveType mean_line() const`
: Returns a curve defining the mean line, $\mathbf{m}(\zeta)$, as a function of the chord fraction ζ. If the curve returned is undefined, there is no cambre.

  The default version of this function defines the mean line as the average of the pressure and suction side curves: $\mathbf{m}(\zeta) = \frac{1}{2}\big(\mathbf{p}(\zeta) + \mathbf{s}(\zeta)\big)$. Note that this curve (the default only) will not have a well-defined derivative at the leading edge.

`ParamCurveType zeta() const`

Returns a curve giving the fractional chord length, ζ, as a function of the curve parameter ξ.

`ParamCurveType xi(bool pressure) const`

Returns a curve giving the curve parameter ξ as a function of the fractional chord length, ζ. If `pressure` is true, the mapping is for the pressure side of the airfoil (ξ will be in the range $[0, \frac{1}{2}]$); otherwise it is for the suction side (ξ will be in the range $[\frac{1}{2}, 1]$).

`bool is_closed() const`

Returns true if the airfoil is closed at the trailing edge.

`bool is_blunt() const`

Returns true if the airfoil is blunt at the trailing edge: i.e. the unit normal at the trailing edge on the suction side is the same as the unit normal on the pressure side. If the trailing edge is not closed, it is not blunt.

`void close_trailing_edge()`

Ensures that the airfoil has a closed trailing edge; does nothing if the trailing edge is already closed. The trailing edge will be sharp.

`void make_trailing_edge_blunt(F radius)`

If the airfoil has an open trailing edge, closes it such that it is blunt; does nothing if the trailing edge is already closed. The argument `radius` is the approximate radius of curvature of the trailing edge. It must be strictly positive.

`bool is_canonical() const`

True if the airfoil is canonical: i.e. its leading edge is at (0,0) and its trailing edge is at (1,0).

`void canonical()`

Scales and rotates the airfoil so that the leading edge is at (0,0) and the trailing edge is at (1,0).

`Point leading_edge() const`

Returns the location of the leading edge.

`Point trailing_edge() const`

Returns the location of the trailing edge.

`F chord_length() const`

Returns the chord length.

`F leading_edge_radius() const`

Returns the radius of curvature at the leading edge. Since the curvature at the leading edge is often discontinuous, it is best to treat the returned value as an approximation.

`F trailing_edge_radius() const`

Returns the radius of curvature at the trailing edge. Throws a `ProgError` if the trailing edge is open or sharp.

`F radius_of_curvature(F xi) const`

Returns the radius of curvature at ξ. Note that, as airfoils often consist of piecewise curves, the curvature may not be continuous.

5 Airfoils defined by specifying the complete airfoil curve

Airfoils can be defined either by specifying the complete airfoil curve, $\mathbf{x}(\xi)$, or by specifying the pressure and suction side curves, $\mathbf{p}(\zeta)$ and $\mathbf{s}(\zeta)$. In this section we consider airfoils which use the former method.

Since the airfoil is defined in terms of the parameter ξ, the parameter ζ must also be defined in terms of ξ. This is done by projecting the average of $\mathbf{x}(\xi)$ and $\mathbf{x}(1-\xi)$ onto the chord line:

$$\zeta(\xi) = \frac{\Big(\frac{1}{2}\big(\mathbf{x}(\xi) + \mathbf{x}(1-\xi)\big) - \mathbf{x}_{le}\Big) \cdot (\mathbf{x}_{te} - \mathbf{x}_{le})}{|\mathbf{x}_{te} - \mathbf{x}_{le}|^2} \tag{2}$$

The inverse of this curve having its value in $[0, \frac{1}{2}]$ (the pressure side) is denoted $\xi_p(\zeta)$:

$$\xi_p\big(\zeta(\xi)\big) = \xi \quad \text{for all } \xi \in [0, \tfrac{1}{2}] \tag{3}$$

Similarly, the inverse of $\zeta(\xi)$ having its value in $[\frac{1}{2}, 1]$ is denoted $\xi_s(\zeta)$. The pressure and suction curves are then defined by

$$\mathbf{p}(\zeta) = \mathbf{x}\big(\xi_p(\zeta)\big); \qquad \mathbf{s}(\zeta) = \mathbf{x}\big(\xi_s(\zeta)\big) \tag{4}$$

Notice that Equation (2) implies that $\zeta(1-\xi) = \zeta(\xi)$ which in turn requires that $\xi_p(\zeta) = 1 - \xi_s(\zeta)$. Therefore the default mean line curve can be written

$$\mathbf{m}(\zeta) = \tfrac{1}{2}\big(\mathbf{p}(\zeta)+\mathbf{s}(\zeta)\big) = \tfrac{1}{2}\big[\mathbf{x}\big(\xi_p(\zeta)\big)+\mathbf{x}\big(1-\xi_p(\zeta)\big)\big] = \tfrac{1}{2}\big[\mathbf{x}\big(1-\xi_s(\zeta)\big)+\mathbf{x}\big(\xi_s(\zeta)\big)\big] \tag{5}$$

Therefore the curve $\frac{1}{2}\big(\mathbf{x}(\xi) + \mathbf{x}(1-\xi)\big)$ also traces out the mean line for ξ in $[0, \frac{1}{2}]$ or $[\frac{1}{2}, 1]$.

5.1 Airfoils defined using offsets

The Airfoil library provides two classes that will define an airfoil by splining a list of (x, y) offsets. In the first a standard cubic spline (see Reference 4, Section 8.3) is used; in the second B-splines are used. The offsets are splined to generate the surface of the airfoil in terms of the parameter ξ. The curves relating ξ to the fractional chord length ζ and the curves giving the pressure and suction sides of the airfoil surface parameterized by ζ are determined from the curve parameterized by ξ.

The offsets should be ordered starting at the trailing edge on the pressure side, proceeding along the pressure side to the leading edge, then back along the suction side to the trailing edge. There must be an offset point at the leading edge; the spline parameterization is adjusted so that this point has a ξ-value of $\frac{1}{2}$.

The `enum TrailingEdgeSpec` is used for specifying the geometry at the trailing edge of airfoils defined using offsets. It can have the following values:

`open_te`
: The trailing edge of the airfoil will be left open.

`sharp_te`
: The trailing edge will be closed but will remain sharp: i.e. the normals at the trailing edge on the pressure and suction sides will differ.

`blunt_te`
: The trailing edge will be closed and the normals at the trailing edge on the pressure and suction sides will be the same.

5.1.1 Offsets splined using a standard cubic spline

The class `OffsetAirfoil<F>` is a specialization of `Airfoil<F>` that defines the airfoil surface by interpolating a list of (x, y) offsets with a standard cubic spline. It has the following members in addition to the default constructor, copy constructor, destructor and assignment operator.

`typedef typename Airfoil<F>::Point Point`
: The type of a point on the airfoil.

`typedef std::vector<Point> OffsetList`
: The type of the list of offsets.

```
OffsetAirfoil(const OffsetList &list, unsigned ile,
              TrailingEdgeSpec te_spec = open_te)
```
Makes an airfoil using the offsets in `list`. The offset point at the leading edge

has index `ile`: i.e. the leading edge point is `list[ile]`.

The offset points are interpolated using a standard cubic spline (see Reference 4, Section 8.3). The argument `te_spec` indicates how the trailing edge will be treated.

If the points at the trailing edge on the pressure and suction sides are the same and if `te_spec` is `blunt_te`, then the cubic spline will be periodic. Otherwise the spline will be a simple cubic spline.

If the points at the trailing edge on the pressure and suction sides differ and if `te_spec` is `sharp_te` or `blunt_te`, then either `close_trailing_edge()` or `make_trailing_edge_blunt()` is called to close the trailing edge.

```
void define(const OffsetList &list, unsigned ile,
        TrailingEdgeSpec te_spec = open_te)
```

Defines the airfoil in a manner similar to the constructor described above.

```
void close_trailing_edge()
```

This function, inherited from `Airfoil<F>`, closes the trailing edge by closing the ξ-curve using a parabola starting at $\zeta_0 = 1 - \frac{1}{2}\Delta$ where Δ is the separation of the points at the trailing edge: $\Delta = |\mathbf{x}(1) - \mathbf{x}(0)|$. A straight line is constructed which passes through the mean line at $\mathbf{m}(\zeta_0)$ and is normal to it. The values of ξ where the line crosses the ξ-curve on the pressure and suction sides will be denoted ξ_p and ξ_s. The ξ-curve, $\mathbf{x}(\xi)$, is then replaced by:

$$\mathbf{x}_{new}(\xi) = \begin{cases} \mathbf{x}(\xi) - \dfrac{(\xi - \xi_p)^2\big(\mathbf{x}(0) - \mathbf{x}_{te}\big)}{\xi_p^2}, & \xi \in [0, \xi_p] \\ \mathbf{x}(\xi), & \xi \in [\xi_p, \xi_s] \\ \mathbf{x}(\xi) - \dfrac{(\xi - \xi_s)^2\big(\mathbf{x}(1) - \mathbf{x}_{te}\big)}{(1 - \xi_s)^2}, & \xi \in [\xi_s, 1] \end{cases} \tag{6}$$

where $\mathbf{x}_{te} = \frac{1}{2}\big(\mathbf{x}(0) + \mathbf{x}(1)\big)$.

```
void make_trailing_edge_blunt(F radius)
```

If the trailing edge is open, this function, inherited from `Airfoil<F>`, closes it such that it is blunt. If the trailing edge is already closed, nothing is done. The algorithm used is described in Annex B.

5.1.2 Offsets splined using B-splines

A `BSplineOffsetAirfoil<F>` is an airfoil defined using a list of offsets which are splined using B-splines with a prescribed knot sequence. Unlike `OffsetAirfoil<F>`, the knots need not lie at the offset points.

The main use of `BSplineOffsetAirfoil<F>` is when several airfoils are to be combined to create a surface. Each airfoil is represented as a `BSplineOffsetAirfoil<F>` with the same knot sequence. The spline coefficients can then be splined to generate a tensor product B-spline surface (see Reference 4, Section 9.2).

Currently it is not possible to make a `BSplineOffsetAirfoil<F>` have a blunt trailing edge.

```
typedef std::vector<Point> OffsetList
```

The type of the list of offsets.

```
typedef typename Spline::BSpline<Point,F>::CoefArray CoefArray
```

The type of the list of B-spline coefficients use by the spline of the airfoil surface as a function of ξ.

```
BSplineOffsetAirfoil(const OffsetList &list, unsigned ile,
                     TrailingEdgeSpec te_spec = open_te)
```

Makes an airfoil by splining the offsets in `list` to obtain the airfoil surface parameterized using ξ. The offset point at the leading edge has index `ile`. The spline will be cubic and its knots will be calculated from the ξ-values of the offset points (determined using `make_xi_values_from_offsets()`) using the not-a-knot condition. The argument `te_spec` indicates how the trailing edge will be treated. Currently `te_spec` is not allowed to be `blunt_te`. If `te_spec` is `sharp_te`, the trailing edge will be closed even if the offset points there differ.

```
BSplineOffsetAirfoil(const OffsetList &list, unsigned ile, unsigned k,
                     const Spline::KnotSeq<F> &kts,
                     TrailingEdgeSpec te_spec = open_te)
```

Makes an airfoil by splining the offsets in `list` to obtain the airfoil surface parameterized using ξ. The offset point at the leading edge has index `ile`. The order of the spline is `k` and its knot sequence is `kts`. The argument `te_spec` indicates how the trailing edge will be treated. Currently `te_spec` is not allowed to be `blunt_te`. If `te_spec` is `sharp_te`, the trailing edge will be closed even if the offset points there differ.

The size of `list` must equal `k` plus the size of `kts`.

```
BSplineOffsetAirfoil(const Spline::KnotSeq<F> &xi_vals,
                     const OffsetList &list,
                     TrailingEdgeSpec te_spec = open_te)
```

Makes an airfoil by splining the offsets in `list` to obtain the airfoil surface parameterized using ξ. The ξ-values of the offset points are in `xi_vals`. The spline will be cubic. The argument `te_spec` indicates how the trailing edge will be treated. Currently `te_spec` is not allowed to be `blunt_te`. If `te_spec` is `sharp_te`, the trailing edge will be closed even if the offset points there differ.

The size of `list` must equal the size of `xi_vals` and also equal `k` plus the size of `kts`.

```
BSplineOffsetAirfoil(const Spline::KnotSeq<F> &xi_vals,
                     const OffsetList &list, unsigned k,
                     const Spline::KnotSeq<F> &kts,
                     TrailingEdgeSpec te_spec = open_te)
```

Makes an airfoil by splining the offsets in `list` to obtain the airfoil surface parameterized using ξ. The ξ-values of the offset points are in `xi_vals`. The order of the spline is `k` and its knot sequence is `kts`. The argument `te_spec` indicates how the trailing edge will be treated. Currently `te_spec` is not allowed to be `blunt_te`. If `te_spec` is `sharp_te`, the trailing edge will be closed even if the offset points there differ.

The size of `list` must equal the size of `xi_vals` and also equal `k` plus the size of `kts`.

```
void define(const OffsetList &list, unsigned ile,
         TrailingEdgeSpec te_spec = open_te)
```

Defines the airfoil in the same manner as the constructor with the same arguments.

```
void define(const OffsetList &list, unsigned ile,
         unsigned k, const Spline::KnotSeq<F> &kts,
         TrailingEdgeSpec te_spec = open_te)
```

Defines the airfoil in the same manner as the constructor with the same arguments.

```
void define(const Spline::KnotSeq<F> &xi_vals, const OffsetList &list,
         TrailingEdgeSpec te_spec = open_te)
```

Defines the airfoil in the same manner as the constructor with the same arguments.

```
void define(const Spline::KnotSeq<F> &xi_vals, const OffsetList &list,
         unsigned k, const Spline::KnotSeq<F> &kts,
         TrailingEdgeSpec te_spec = open_te)
```

Defines the airfoil in the same manner as the constructor with the same arguments.

```
unsigned order() const
```

Returns the order of the spline of the airfoil surface parameterized using ξ.

`const typename Spline::KnotSeq<F>& knots() const`

Returns the knot sequence used by the airfoil surface parameterized using ξ.

`const CoefArray& Bspline_coefs() const`

Returns the B-spline coefficients used by the airfoil surface parameterized using ξ.

5.2 Joukowski airfoils

Joukowski airfoils are important because there is an analytic solution to the potential flow around them. This makes them suitable for verification of software calculating potential flow.

5.2.1 The Joukowski conformal mapping

A Joukowski airfoil is defined in the complex plane by the conformal mapping $w(z)$ defined by:

$$q(z) = C + (a - C)z; \qquad C = x + iy \tag{7}$$

$$p(z) = q(z) + \frac{a^2}{q(z)} \tag{8}$$

$$w(z) = A + Bp(z); \qquad A = \frac{1}{2}\left(1 + \frac{x^2}{(a-x)^2}\right); \qquad B = \frac{a-2x}{4(a-x)^2} \tag{9}$$

where a is real. The mapping from z to q shifts the unit circle to a new circle centred at C and passing through $(a, 0)$. The mapping from q to p then transforms this circle to an airfoil shape with trailing edge at $(2a, 0)$. The mapping from p to w then scales and translates the airfoil so that it is canonical. Each of the mappings is illustrated in Figure 2.

The surface of the airfoil can be parameterized with the angle θ used to traverse the unit circle: $z = e^{i\theta}$. The trailing edge is at $\theta = 0$ or 2π and the leading edge is at

$$\theta_{le} = \pi + 2\phi; \qquad \phi = \arctan\left(\frac{y}{a-x}\right); \qquad z_{le} = \frac{\overline{C} - a}{a - C} \tag{10}$$

The thickness of the airfoil is controlled by x. When it is small, the maximum thickness occurs when $\theta = 2\pi/3$ and is given by:

$$t = -\frac{3\sqrt{3}x}{4a} \tag{11}$$

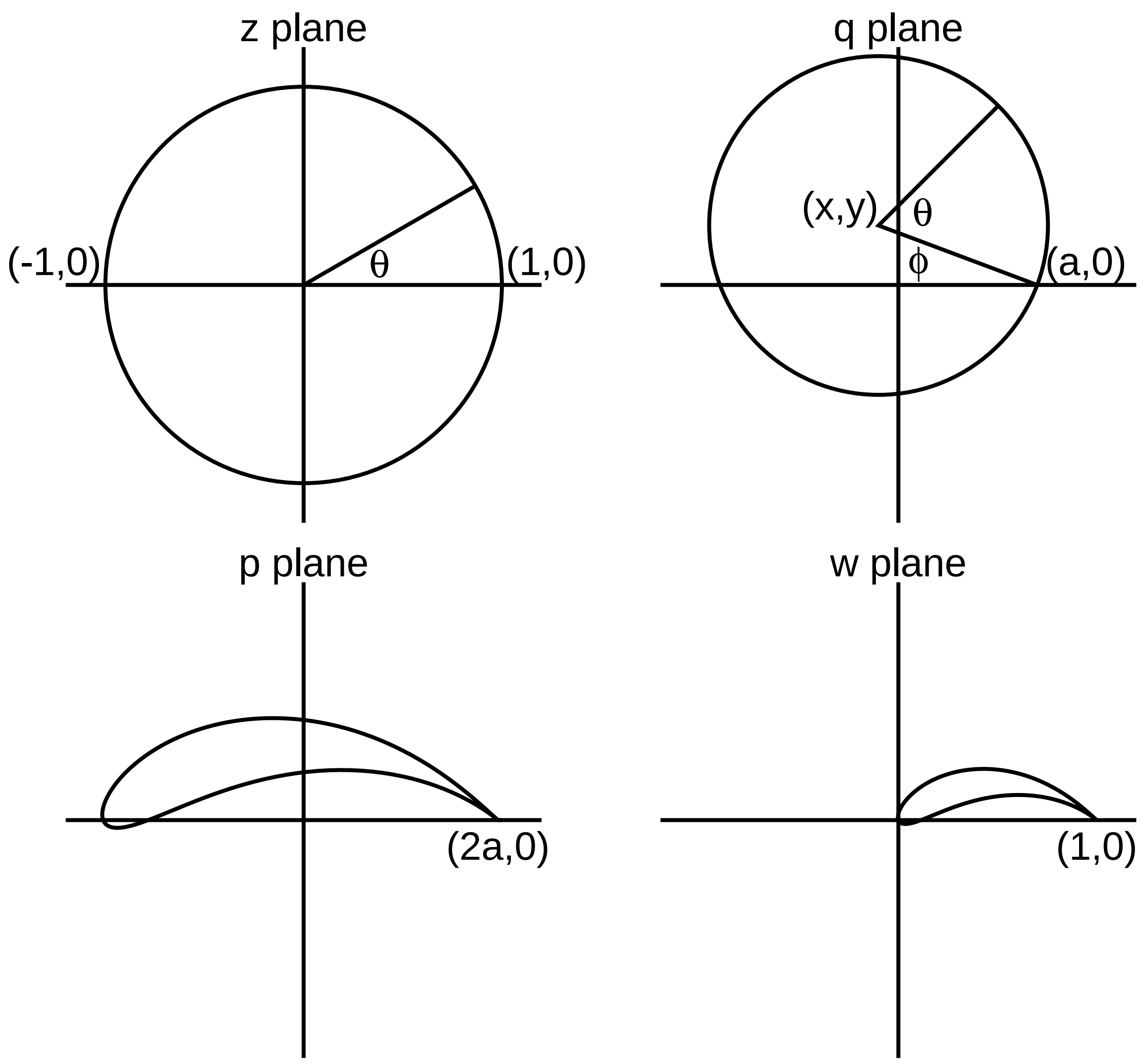

Figure 2: *The Joukowski mappings.*

As x approaches zero, the airfoil becomes infinitely thin. Therefore we can define a mean line by setting $x = 0$:

$$w_m(\theta) = \frac{(a+q)^2}{4aq}; \qquad q = iy + (a - iy)e^{i\theta} \tag{12}$$

The cambre is determined from the point where $\mathrm{Im}(dw_m/d\theta) = 0$. To first order in y

$$c = \frac{y}{2a} \tag{13}$$

We can specify a canonical airfoil uniquely given c and t by setting $a = 1$ and using the first order expressions of Equations (11) and (13).

$$x = -\frac{4t}{3\sqrt{3}}; \qquad y = 2c \tag{14}$$

The class `JoukowskiMapping<F>` represents the conformal mapping $w(z)$ from the region in the complex plane exterior to the unit circle, to the region exterior to a Joukowksi airfoil in the complex plane. The airfoil is canonical: i.e. the leading and trailing edges are at $w = 0$ and $w = 1$ respectively.

`JoukowskiMapping<F>` is derived from `CurveLib::Curve<1U,Cmplx,Cmplx>`, where `Cmplx` is an alias for `std::complex<F>`: i.e. `JoukowskiMapping<F>` is a curve with a complex argument returning a complex value. It has the following member functions the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`JoukowskiMapping(F t, F c)`
: Makes a Joukowski mapping for an airfoil with thickness `t` and cambre `c`. The thickness and cambre are used to define x and y using the first order expressions of Equation (14); therefore the thickness and cambre of the airfoil will only approximate the values of `t` and `c`.

`void define(F t, F c)`
: Redefines the airfoil in the same manner as the constructor above.

`F cambre() const`
: Returns the cambre used when defining the mapping.

`F thickness() const`
: Returns the thickness used when defining the mapping.

`FloatType leading_edge() const`
: Returns the z-value of the leading edge.

It is often also convenient to be able to evaluate the inverse Joukowski mapping: i.e. evaluate z given w:

$$p(w) = \frac{w - A}{B}; \quad q(w) = \tfrac{1}{2}\bigl(p(w) \pm \sqrt{p(w)^2 - 4a^2}\,\bigr); \quad z(w) = \frac{q(w) - C}{a - C} \tag{15}$$

In the evaluation of $q(w)$, the root which makes $|z(w)| > 1$ must be chosen.

This mapping is implemented by the class `InvJoukowskiMapping<F>` which has constructors and `define` function similar to `JoukowskiMapping<F>`. It also has the following constructor:

`InvJoukowskiMapping(JoukowskiMapping<F> jm)`
 Makes a mapping which is the inverse of `jm`.

5.2.2 A class to represent a Joukowski airfoil

The class `JoukowskiAirfoil<F>` represents a Joukowski airfoil. It is derived from `Airfoil<F>`.

The airfoil surface can be traced using a Joukowski mapping with $z = e^{i\theta}$. To conform with the parameterization of the `Afoil` classes, θ must be converted to a new parameter whose value is 0 or 1 at the trailing edge and $\frac{1}{2}$ at the leading edge. This can be done using the mapping:

$$\theta = -2\pi\xi + 32\phi\xi^2(1 - \xi)^2 \tag{16}$$

Notice that as ξ increases from 0 to 1, θ decreases from 0 to -2π, thus traversing the airfoil clockwise in conformance with the requirements for the parameter ξ.

We use $\xi^2(1-\xi)^2$ in Equation (16), rather than $\xi(1-\xi)$, so that the $du/d\eta$ is the same on both sides of the airfoil at the trailing edge. This makes the mean line defined by $\frac{1}{2}\bigl(u(\xi) + u(1 - \xi)\bigr)$ more nearly equidistant between the two sides of the airfoil near the trailing edge.

The parameter ξ defined in Equation (16) has the drawback that $du/d\xi = 0$ at the trailing edge. This might cause problems, for example when generating a normal to the airfoil at the trailing edge. We can avoid this problem by defining

$$\theta = -2\pi\eta + 32\phi\eta^2(1 - \eta^2); \qquad \eta = \tfrac{1}{2}\bigl(1 + \sqrt{\xi} - \sqrt{1 - \xi}\,\bigr) \tag{17}$$

`JoukowskiAirfoil<F>` has the following members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`JoukowskiAirfoil(F t, F c, bool finite_te_deriv = false)`

Makes a Joukowski airfoil with thickness `t` and cambre `c`. The thickness and cambre are used to define x and y using the first order expressions of Equation (14); therefore the thickness and cambre of the airfoil will only approximate the values of `t` and `c`.

If `finite_te_deriv` is true, then the ξ-parameterization defined by Equation (17) is used to ensure that the first derivative of the airfoil ξ-curve will be finite at the trailing edge. If `finite_te_deriv` is false, then the parameterization defined by Equation (16) is used and the first derivative at the trailing edge will be zero. The ξ-curve with finite derivatives evaluates somewhat less efficiently than the one with zero derivatives; however, it is better behaved when, for example, one wishes to generate a normal to the airfoil at the trailing edge.

`void define(F t, F c, bool finite_te_deriv = false)`

Redefines the airfoil in the same manner as the constructor above.

5.2.3 Classes to represent the potential flow around a Joukowski airfoil

The namespace `Afoil` also includes classes that can be used to determine the potential flow around a Joukowski airfoil. This is an important verification case for flow solvers.

The complex potential for the flow past a Joukowski airfoil with speed V at infinity at angle of attack α is:

$$\Phi = \frac{V(a-2x)|a-C|}{4(a-x)^2}\left[\left(ze^{-i(\alpha+\phi)} + e^{i(\alpha+\phi)}/z\right) + 2i\sin(\alpha+\phi)\ln\left(\frac{z}{a}\right)\right] \tag{18}$$

The complex velocity is obtained by differentiating the potential with respect to w:

$$v_x - iv_y = \frac{d\Phi}{dw} = \frac{d\phi}{dz}\Bigg/\frac{dw}{dz} \tag{19}$$

Special care must be taken when evaluating the complex velocity at the trailing edge since $dw/dz = 0$ there.

The classes `JoukowskiPotential<F>` and `JoukowskiVelocity<F>` are curves with complex arguments which evaluate the complex potential and complex velocity respectively for the flow past a Joukowski airfoil. They are both derived from the class `Curve<1U,std::complex<F>,std::complex<F> >` in namespace `CurveLib`.

The parameter of these curves is the value of z used in the Joukowski mapping: it is restricted to values outside the unit circle in the complex plane. The point at which the potential is evaluated is $w(z)$, where w is the Joukowski mapping.

`JoukowskiPotential<F>` has the following members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes. The members of `JoukowskiVelocity<F>` are identical: only the returned value is different.

`JoukowskiPotential(F t, F c, F v, Angle<F> alpha)`

Makes a curve which evaluates the potential for an airfoil with thickness `t` and cambre `c`. The flow at infinity is `v` and the angle of attack is `alpha`.

The thickness and cambre are used to define x and y using the first order expressions of Equation (14); therefore the thickness and cambre of the airfoil will only approximate the values of `t` and `c`.

`void define(F t, F c, F v, Angle<F> alpha)`

Redefines the airfoil in the same manner as the constructor above.

6 Airfoils defined by specifying pressure and suction side curves

If an airfoil is defined by specifying the pressure and suction side curves, then the parameter ξ must be defined in terms of ζ and the complete airfoil curve, $\mathbf{x}(\xi)$, must be defined from $p(\zeta)$ and $s(\zeta)$. We first consider the mapping from ζ to ξ.

The pressure and suction side curves are sampled at an initial set of ζ values, ζ_i, for $i \in [1, N]$. One of the sample points, $i = i_{le}$ is chosen to be at the leading edge. Straight line segments between the sampled points are used to approximate the arclength between each adjacent pair of points. For each i, a new point is sampled at $\zeta_{i+1/2} = \frac{1}{2}(\zeta_i + \zeta_{i+1})$. Suppose that these points lie on the pressure side. Then if

$$\left|\mathbf{p}(\zeta_{i+1/2}) - \mathbf{p}(\zeta_i)\right| + \left|\mathbf{p}(\zeta_{i+1}) - \mathbf{p}(\zeta_{i+1/2})\right| > \left|\mathbf{p}(\zeta_{i+1}) - \mathbf{p}(\zeta_i)\right| + \epsilon|\mathbf{x}_{te} - \mathbf{x}_{le}| \qquad (20)$$

the point $\mathbf{p}(\zeta_{i+1})$ is added to the sample set; otherwise the straight line is deemed a sufficiently accurate approximation to the segment. The factor ϵ is set to 10^{-4}. The points on the suction side are refined in a similar way.

Once every segment is sufficiently accurate, the lengths of the segments are used to approximate the fractional arclength, a_i, at each sample point. We define ξ as a function of a by

$$\xi = a + \frac{\left(\frac{1}{2} - a_{le}\right)a^2(1-a)^2}{a_{le}^2(1-a_{le})^2} \qquad (21)$$

where a_{le} is the value of a at the leading edge. This definition ensures that $\xi(0) = 0$, $\xi(a_{le}) = \frac{1}{2}$ and $\xi(1) = 1$ as required.

A Hermite spline (see Reference 4, Section 8) is used to interpolate the ζ_i with respect to the $\xi_i = \xi(a_i)$ yielding the curve $\zeta(\xi)$. The slopes of the spline are first calculated so that the second derivatives of the spline are continuous at the knots (a standard cubic spline). The slope at the leading edge is then adjusted to be exactly zero. The algorithm of Fritsch and Carlson [3] is then used to ensure that the spline is monotonic on each of the pressure and suction sides.

The complete airfoil curve is defined by

$$\mathbf{x}(\xi) = \begin{cases} \mathbf{p}\big(\zeta(\xi)\big) & \text{for } \xi \in [0, \frac{1}{2}] \\ \mathbf{s}\big(\zeta(\xi)\big) & \text{for } \xi \in [\frac{1}{2}, 1] \end{cases} \tag{22}$$

Typically the pressure and suction side curves have a form similar to

$$\mathbf{p}(\zeta) = -\hat{n}\sqrt{2r\zeta} + o(\zeta); \qquad \mathbf{s}(\zeta) = \hat{n}\sqrt{2r\zeta} + o(\zeta) \tag{23}$$

near the leading edge where r is the radius of curvature at the leading edge. From the spline expansion of $\zeta(\xi)$ we have

$$\zeta = \begin{cases} a(\xi - \frac{1}{2})^2 + o(\xi^3) & \text{for } \xi < \frac{1}{2} \\ b(\xi - \frac{1}{2})^2 + o(\xi^3) & \text{for } \xi > \frac{1}{2} \end{cases} \tag{24}$$

where a and b are constants whose values are generally different. This reflects the fact that, by setting the slope to zero at the leading edge, we have altered the spline so that its second derivative is no longer continuous there. Substituting Equations (24) and (23) into Equation (22) gives

$$\mathbf{x}(\xi) = \begin{cases} \sqrt{2ra}\,(\xi - \frac{1}{2})\hat{n} + o\big((\xi - \frac{1}{2})^2\big) & \text{for } \xi < \frac{1}{2} \\ \sqrt{2rb}\,(\xi - \frac{1}{2})\hat{n} + o\big((\xi - \frac{1}{2})^2\big) & \text{for } \xi > \frac{1}{2} \end{cases} \tag{25}$$

Since, in general, $a \neq b$, the curve $\mathbf{x}(\xi)$ will have discontinuous slope at the leading edge. Since we want $\mathbf{x}(\xi)$ to be smooth, it is necessary to adjust $\zeta(\xi)$ at the leading edge so that its second derivative is continuous. This is done as follows.

Let t_{-1} and t_1 be the spline knots for the curve $\zeta(\xi)$ that lie on either side of the leading edge. The values of a and b can be determined from the spline coefficients. We define a new function $\zeta(\xi)$ by

$$\zeta_{new}(\xi) = \zeta(\xi) + \begin{cases} \dfrac{(b-a)(\xi - \frac{1}{2})^2(\xi - t_{-1})^3}{2(\frac{1}{2} - t_{-1})^3} & \text{for } \xi \in [t_{-1}, \frac{1}{2}] \\ \dfrac{(a-b)(\xi - \frac{1}{2})^2(\xi - t_1)^3}{2(\frac{1}{2} - t_1)^3} & \text{for } \xi \in [\frac{1}{2}, t_1] \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

The new function is smooth and has continuous second derivatives.

There is still the problem that because $\mathbf{p}'(\frac{1}{2})$ and $\mathbf{s}'(\frac{1}{2})$ are not well-defined, $\mathbf{x}(\xi)$ as defined by Equation (22) will have undefined derivatives when evaluated at the leading edge, even though the derivatives are well-defined in the limit $\xi \to \frac{1}{2}$. To avoid this problem we use the representation of $\mathbf{x}(\xi)$ in Equation (22) except in a very small region close to the leading edge where we use two parabolic segments instead. The parabolic segments cover the regions $\xi \in [\frac{1}{2}-\delta, \frac{1}{2}]$ and $\xi \in [\frac{1}{2}, \frac{1}{2}+\delta]$ with $\delta = \epsilon^{3/4}$, where ϵ is the smallest floating point number of type `F` such that $1+\epsilon$ is indistinguishable from 1. The coefficients of the parabolae are chosen so that they interpolate the points $\mathbf{p}_0 = \mathbf{p}\big(\zeta(\frac{1}{2}-\delta)\big)$, $\mathbf{p}_1 = \mathbf{p}\big(\zeta(\frac{1}{2}-\frac{1}{2}\delta)\big)$ and $\mathbf{p}_2 = \mathbf{p}\big(\zeta(\frac{1}{2})\big)$ on the pressure side and $\mathbf{s}_0 = \mathbf{s}\big(\zeta(\frac{1}{2})\big)$, $\mathbf{s}_1 = \mathbf{s}\big(\zeta(\frac{1}{2}+\frac{1}{2}\delta)\big)$ and $\mathbf{s}_2 = \mathbf{s}\big(\zeta(\frac{1}{2}+\delta)\big)$ on the suction side. We now define the complete airfoil curve by

$$\mathbf{x}(\xi) = \begin{cases} \mathbf{p}\big(\zeta(\xi)\big) & \text{for } \xi \in [0, \frac{1}{2}-\delta] \\ \mathbf{p}_0 + \dfrac{(3\mathbf{p}_0 - 4\mathbf{p}_1 + \mathbf{p}_2)\big(\xi - \frac{1}{2}\big)}{\delta} & \\ \qquad + \dfrac{(2\mathbf{p}_0 - 4\mathbf{p}_1 + 2\mathbf{p}_2)\big(\xi - \frac{1}{2}\big)^2}{\delta^2} & \text{for } \xi \in [\frac{1}{2}-\delta, \frac{1}{2}] \\ \mathbf{s}_0 - \dfrac{(3\mathbf{s}_0 - 4\mathbf{s}_1 + \mathbf{s}_2)\big(\xi - \frac{1}{2}\big)}{\delta} & \\ \qquad + \dfrac{(2\mathbf{s}_0 - 4\mathbf{s}_1 + 2\mathbf{s}_2)\big(\xi - \frac{1}{2}\big)^2}{\delta^2} & \text{for } \xi \in [\frac{1}{2}, \frac{1}{2}+\delta] \\ \mathbf{s}\big(\zeta(\xi)\big) & \text{for } \xi \in [\frac{1}{2}+\delta, 1] \end{cases} \tag{27}$$

If the trailing edge is blunt, a similar process is used on the regions $\xi \in [0, \delta]$ and $\xi \in [1-\delta, 1]$.

The curve $\mathbf{x}(\xi)$ is not smooth at $\xi = \frac{1}{2}-\delta$, $\frac{1}{2}$ and $\frac{1}{2}+\delta$, but the discontinuities are so small (because δ is so small) that for all practical purposes this is inconsequential.

6.1 Airfoils defined using a thickness curve and a mean line offset curve

A common method of defining airfoils is via a thickness curve (also known as a thickness distribution) and a mean line offset curve. Define the thickness curve $t(\zeta)$ to be a non-negative scalar-valued function such that $t(0) = 0$ and $t(\zeta) > 0$ for all ζ in (0,1). The value of $t(\zeta)$ is the half-thickness of the airfoil at ζ. Also define a mean line offset curve $m(\zeta)$ to be a scalar-valued function giving the y-offset of the mean line from the chord line. The two sides of the airfoil are defined by:

$$\mathbf{p}(\zeta) = \mathbf{m}(\zeta) - \hat{n}(\zeta)t(\zeta) \tag{28}$$

$$\mathbf{s}(\zeta) = \mathbf{m}(\zeta) + \hat{n}(\zeta)t(\zeta) \tag{29}$$

$$\mathbf{m}(\zeta) = \big(x_{le} + \zeta(x_{te} - x_{le})\big)\hat{x} + m(\zeta)\hat{y} \tag{30}$$

where $\mathbf{m}(\zeta)$ is the mean line (note that this is different from the mean line offset) and $\hat{n} = (n_x, n_y)$ is a unit normal to the mean line:

$$n_x = -\frac{m'(\zeta)}{\sqrt{m'(\zeta)^2 + (x_{te} - x_{le})^2}}; \qquad n_y = \frac{x_{te} - x_{le}}{\sqrt{m'(\zeta)^2 + (x_{te} - x_{le})^2}} \tag{31}$$

The airfoil is canonical if $x_{le} = 0$, $x_{te} = 1$ and $m(0) = m(1) = 0$.

Notice that because the normal depends on the derivative of the mean line offset curve, $\mathbf{p}(\zeta)$ and $\mathbf{s}(\zeta)$ will have one degree less continuity than $m(\zeta)$. In particular, if $m(\zeta)$ is only C^1, then $\mathbf{p}(\zeta)$ and $\mathbf{s}(\zeta)$ will only be C^0.

`ThickDistAirfoil<F>` is a base class for airfoils which are defined using a thickness curve and a mean line offset curve represented by the classes `ThicknessCurve<F>` and `MeanLineOffset<F>` respectively: see Sections 6.1.1 and 6.1.2. It is derived from the base class `Airfoil<F>` and has the following members as well as the default and copy constructors, virtual destructor, assignment operator and those inherited from its base classes.

```
ThickDistAirfoil(ThicknessCurve<F> t, MeanLineOffset<F> m,
                 F xle = F(0), F xte = F(1))
```

Makes an airfoil with thickness curve `t` and mean line offset `m`. The x-values of the leading and trailing edges are given by `xle` and `xte`.

```
MeanLineOffset<F> mean_line_offset() const
```

Returns the mean line offset curve, $m(\zeta)$. If the returned curve is undefined (i.e. if `mean_line_offset().is_defined()` returns `false`), then the airfoil has no cambre.

Note that the returned curve is scalar-valued unlike the curve returned by the inherited function `mean_line()` whose value is a 2-vector representing the (x, y) coordinates of a point on the mean line.

```
void set_mean_line_offset(MeanLineOffset<F> ml)
```

Sets the mean line offset curve to `ml`. If `ml` is undefined (i.e. if `ml.is_defined()` is false), the airfoil will have no cambre.

```
ThicknessCurve<F> thickness() const
```

Returns a curve defining the thickness curve, $t(\zeta)$.

```
void set_thickness(ThicknessCurve<F> t, bool blunt = false)
```

Sets the thickness curve to `t`. If `blunt` is true, then the airfoil will assume that the thickness curve is closed and blunt.

`void close_trailing_edge_by_extension()`

Ensures that the airfoil has a closed trailing edge by extending the range of the thickness curve until it is zero; the parameter ζ is then scaled so that it is 1 at the new trailing edge location: see Section 6.1.2 for a description of the algorithm used. The trailing edge will be sharp.

An `Error` will be thrown if the slope of the thickness curve is not strictly negative at $\zeta = 1$.

`void close_trailing_edge_with_parabola(F zeta0 = F(1))`

Ensures that the thickness curve has a closed trailing edge by adding a parabolic curve to it starting at $\zeta =$ `zeta0`. The parabola is chosen so that the thickness curve is smooth: see Section 6.1.2 for a description of the algorithm used. The new trailing edge is sharp.

If the value of `zeta0` is not in [0,1) the location of maximum thickness is used.

If the trailing edge is already closed, nothing is done.

`void close_trailing_edge()`

This function, inherited from `Airfoil<F>`, closes the trailing edge by closing the thickness curve using a parabola starting at $\zeta = 1 - \frac{1}{2}\Delta$ where Δ is the thickness at the trailing edge: i.e. if `afoil` is a `ThickDistAirfoil<F>`,

```
afoil.close_trailing_edge();
```

is equivalent to:

```
F half_delta = afoil.thickness()(1);
afoil.close_trailing_edge_with_parabola(1-half_delta);
```

6.1.1 Classes to represent mean line offset curves

An important class of airfoils defines the airfoil shape using a mean line offset curve and a thickness curve: see Section 6.1. In this section we describe classes to represent mean line offset curves.

A mean line offset curve, $m(\zeta)$, is said to be *canonical* if its value at both $\zeta = 0$ and $\zeta = 1$ is zero. This is a necessary condition for the airfoil using the mean line offset curve to be canonical.

The value of the maximum offset of the mean line is the cambre, c. The location of the maximum offset is denoted ζ_m.

For thin airfoils, the mean line offset curve is sufficient to define the characteristics of the airfoil lift (see, for example, Abbot and von Doenhoff [2], Chapter 4). The lift coefficient predicted by thin airfoil theory for a canonical meanline offset curve is

$$C_L = 2\pi(\alpha - \alpha_0); \qquad \alpha_0 = -\frac{1}{\pi}\int_0^1 m(\zeta) f_1(\zeta)\, d\zeta; \qquad f_1(\zeta) = \frac{1}{\sqrt{\zeta(1-\zeta)^3}} \tag{32}$$

where α_0 is the angle of zero lift. The pitching moment around $\zeta = \frac{1}{4}$ is

$$C_m = 2\int_0^1 m(\zeta) f_2(\zeta)\, d\zeta + \frac{\pi}{2}\alpha_0; \qquad f_2(\zeta) = \frac{1-2\zeta}{\sqrt{\zeta(1-\zeta)}} \tag{33}$$

When $\alpha = \alpha_i$, with

$$\alpha_i = \frac{1}{2\pi}\int_0^1 m(\zeta) f_3(\zeta)\, d\zeta; \qquad f_3(\zeta) = \frac{1-2\zeta}{\sqrt{\zeta^3(1-\zeta)^3}} \tag{34}$$

the velocity field near the leading edge of an infinitesimally thin airfoil has no singularities; α_i is called the ideal angle of attack. The lift coefficient at the ideal angle of attack is called the ideal or design lift coefficient, C_{Li}.

Notice that $f_2(\zeta)$ and $f_3(\zeta)$ are antisymmetric about $\zeta = \frac{1}{2}$. Therefore, if the mean line offset is symmetric about $\zeta = \frac{1}{2}$, the integrals in Equations (33) and (34) vanish and we have

$$\alpha_i = 0; \qquad C_{Li} = -2\pi\alpha_0; \qquad C_m = \frac{\pi\alpha_0}{2} = -\frac{C_{Li}}{4} \tag{35}$$

Mean line offset curves are represented by the class `MeanLineOffset<F>` derived from the base class `CurveLib::Curve<1U,F,F>`. It has the following public members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`typedef CurveLib::Curve<1U,F,F> CurveType`

The type of the curve used to represent the mean line offset curve.

`MeanLineOffset()`

Makes a canonical mean line offset curve having no cambre.

`MeanLineOffset(CurveType crv)`

Makes a mean line offset curve using the curve `crv`.

`void define(CurveType crv)`

Redefines the mean line offset curve using the curve `crv`. If `crv` is undefined, the mean line will have no cambre.

`F cambre() const`
: Returns the cambre.

`bool is_canonical() const`
: True if the mean line offset curve is canonical.

`F zeta_at_maximum_ordinate() const`
: Returns the location of the maximum ordinate.

`F design_lift_coefficient() const`
: Returns the design lift coefficient. The mean line offset curve must be canonical.

`Angle<F> ideal_angle_of_attack() const`
: Returns the ideal angle of attack. The mean line offset curve must be canonical.

`Angle<F> angle_of_zero_lift() const`
: Returns the angle of zero lift. The mean line offset curve must be canonical.

`F pitching_moment_coefficient() const`
: Returns the pitching moment coefficient about $\zeta = \frac{1}{4}$. The mean line offset curve must be canonical.

When the default constructor is used, the mean line offset curve will be defined to be canonical with no cambre. This paradigm extends to the classes derived from `MeanLineOffset<F>`.

Due to its implementation as a handle to an underlying representation (inherited from the base class `CurveLib::Curve<1U,F,F>`), the member functions of the class `MeanLineOffset<F>` are inherently polymorphic, even though they are not virtual. Therefore, for example, the code

```
using namespace Afoil;
MeanLineOffset<double> afoil = NACA4DigitMeanLineOffset(0.4,0.03);
double alpha_i = afoil.ideal_lift_coefficient();
```

will cause `alpha_i` to be evaluated using the functions appropriate to a NACA 4-digit mean line (see Section 6.1.1.2), with a result which has machine accuracy.

6.1.1.1 Mean line offset curves constructed from piecewise polynomial splines

`PPSplineMeanLineOffset<F>` is a mean line offset curve which is defined using a piecewise polynomial spline (see the description of the class `Spline::PPSpline<F,F>` in Reference 4, Section 6). Several of the commonly used mean line offset curves can be represented in this way. Moreover, the angle of zero lift, ideal angle of attack, design lift coefficient and the pitching moment can all be calculated exactly (according to thin airfoil theory) for these mean lines: see Annex A.

`PPSplineMeanLineOffset<F>` is derived from the class `MeanLineOffset<F>` and has the following public members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`typedef Spline::PPSpline<F,F> SplineType`
: The type of the curve used to represent the mean line offset curve.

`PPSplineMeanLineOffset(SplineType s)`
: Makes a mean line offset curve using the spline `s`.

`void define(SplineType s)`
: Redefines the mean line offset curve using the spline `s`.

6.1.1.2 NACA 4-digit mean line offset curve

The mean line offset curves used by the NACA 4-digit series of airfoils (see Abbot and von Doenhoff [2], Chapter 6.4) have the form:

$$m(\zeta) = \begin{cases} \dfrac{c\zeta(2\zeta_m - \zeta)}{\zeta_m^2} & \text{for } \zeta \in [0, \zeta_m] \\ \dfrac{c(1-\zeta)(1-2\zeta_m+\zeta)}{(1-\zeta_m)^2} & \text{for } \zeta \in [\zeta_m, 1] \end{cases} \tag{36}$$

A typical value for ζ_m is 0.4.

The ideal angle of attack, the angle of zero lift and the design lift coefficient are given by

$$\alpha_i = \frac{c(1-2\zeta_m)}{2\pi\zeta_m^2(1-\zeta_m)^2}\left[4\sqrt{\zeta_m(1-\zeta_m)} - 2\pi\zeta_m^2 - 2(1-2\zeta_m)\arccos(1-2\zeta_m)\right] \tag{37}$$

$$\alpha_0 = -\frac{c}{4\pi\zeta_m^2(1-\zeta_m)^2}\Big[4(1-2\zeta_m)(2\zeta_m-3)\sqrt{\zeta_m(1-\zeta_m)} \\ - 2(4\zeta_m-3)\big(\pi\zeta_m^2 + (1-2\zeta_m)\arccos(1-2\zeta_m)\big)\Big] \tag{38}$$

$$C_{Li} = \frac{c}{2\zeta_m^2(1-\zeta_m)^2}\Big[-4(1-2\zeta_m)^2\sqrt{\zeta_m(1-\zeta_m)} \\ + 2\pi\zeta_m^2 + 2(1-2\zeta_m)\arccos(1-2\zeta_m)\Big] \tag{39}$$

$$C_m = \frac{c}{12\zeta_m^2(1-\zeta_m)^2}\Big[2(4\zeta_m^2-1)(4\zeta_m-3)\sqrt{\zeta_m(1-\zeta_m)} \\ - 3\zeta_m^2\pi - 3(1-2\zeta_m)\arccos(1-2\zeta_m))\Big] \tag{40}$$

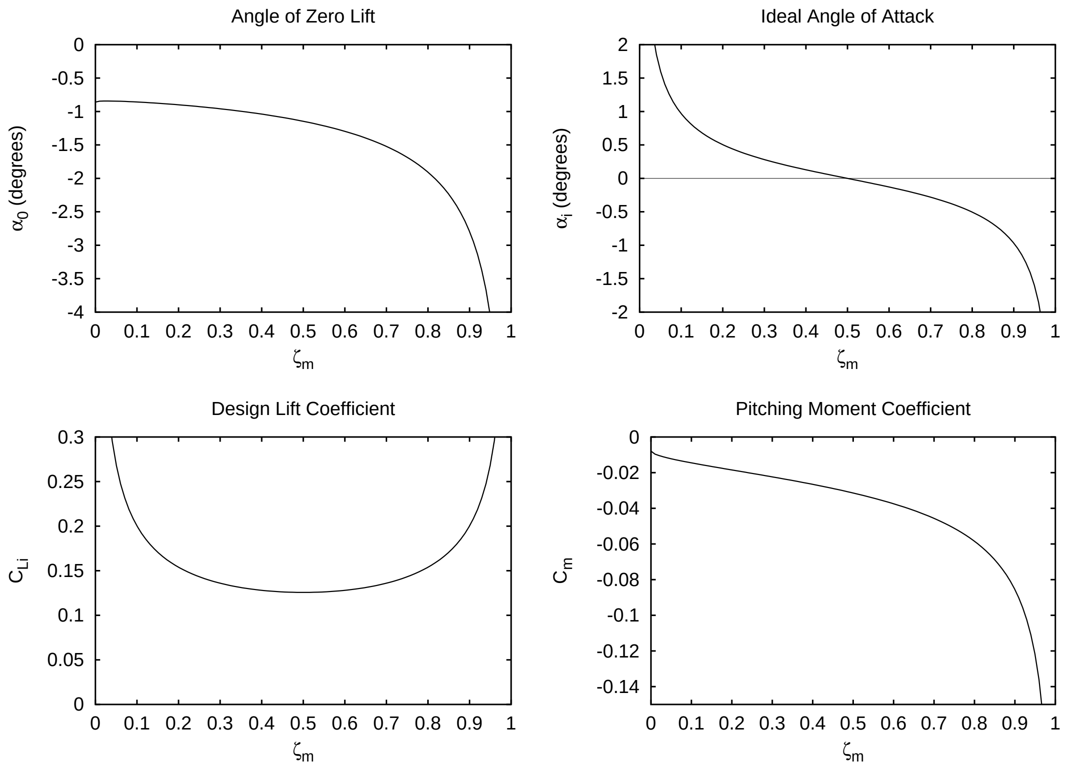

Figure 3: *The lift characteristics as a function of ζ_m for a NACA 4-digit airfoil with cambre $c = 0.01$.*

Figure 3 plots α_0, α_i, C_{Li} and C_m as a function of ζ_m for a NACA 4-digit airfoil with cambre equal to 1% of chord.

The class `NACA4DigitMeanLineOffset<F>` represents NACA 4-digit mean line offset curves. It is derived from the base class `PPSplineMeanLineOffset<F>` and has the following has the following public members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`NACA4DigitMeanLineOffset(F zetam, F c)`
Makes a NACA 4-digit mean line offset curve for an airfoil having cambre `c`. The maximum ordinate occurs when ζ is `zetam`.

`NACA4DigitMeanLineOffset(const Str &desig)`
Makes a NACA 4-digit mean line offset curve having designation `desig`. The string `desig` has two digits, the first giving the cambre in percent of the chord, the second giving the location of the maximum ordinate in tenths of the chord. For example, a designation of `"34"` indicates that $c = 0.03$ and $\zeta_m = 0.4$.

`void set_cambre(F c)`

Redefines the mean line offset curve to have cambre `c`.

`void set_zeta_at_maximum_ordinate(F zetam)`

Sets the location of the maximum ordinate to `zetam`.

`void define(F zetam, F c)`

Redefine the mean line offset curve to have maximum ordinate at `zetam` and cambre `c`.

6.1.1.3 NACA 5-digit mean line offset curve

The mean line offset curves used by the NACA 5-digit series of airfoils (see Abbot and von Doenhoff [2], Chapter 6.5) have the form:

$$m(\zeta) = \begin{cases} \frac{1}{6}k_1\zeta\big(\zeta^2 - 3\zeta_j\zeta + \zeta_j^2(3-\zeta_j)\big) & \text{for } \zeta \in [0,\zeta_j] \\ \frac{1}{6}k_1\zeta_j^3(1-\zeta) & \text{for } \zeta \in [\zeta_j, 1] \end{cases} \tag{41}$$

The location of the maximum ordinate occurs forward of ζ_j at $\zeta_m = \zeta_j - \sqrt{\zeta_j^3/3}$ which can be inverted to give ζ_j as a function of ζ_m,

$$\zeta_j = 1 - 2\sqrt{1-2\zeta_m}\,\sin\left(\pi/6 - \frac{1}{3}\arctan\left(\frac{\zeta_m\sqrt{\zeta_m(4-9\zeta_m)}}{2-6\zeta_m+3\zeta_m^2}\right)\right) \tag{42}$$

Since ζ_j cannot exceed 1, ζ_m cannot exceed $1 - 1/\sqrt{3} \approx 0.42265$.

The cambre is

$$c = \frac{k_1\zeta_j^3}{6}\left(1 - \zeta_j + \frac{2\sqrt{3\zeta_j^3}}{9}\right) \tag{43}$$

and the ideal angle of attack, the angle of zero lift and the design lift coefficient are given by

$$\alpha_i = \frac{k_1}{48\pi}\Big[-18(1-2\zeta_j)\sqrt{\zeta_j(1-\zeta_j)} - 8\pi\zeta_j^3 + (24\zeta_j^2 - 24\zeta_j + 9)\arccos(1-2\zeta_j)\Big] \tag{44}$$

$$\alpha_0 = -\frac{k_1}{48\pi}\Big[(16\zeta_j^2 - 52\zeta_j + 30)\sqrt{\zeta_j(1-\zeta_j)} + 8\pi\zeta_j^3 - (24\zeta_j^2 - 36\zeta_j + 15)\arccos(1-2\zeta_j)\Big] \tag{45}$$

$$C_{Li} = \frac{k_1}{12}\Big[(8\zeta_j^2 - 8\zeta_j + 6)\sqrt{\zeta_j(1-\zeta_j)} - 3(1-2\zeta_j)\arccos(1-2\zeta_j)\Big] \tag{46}$$

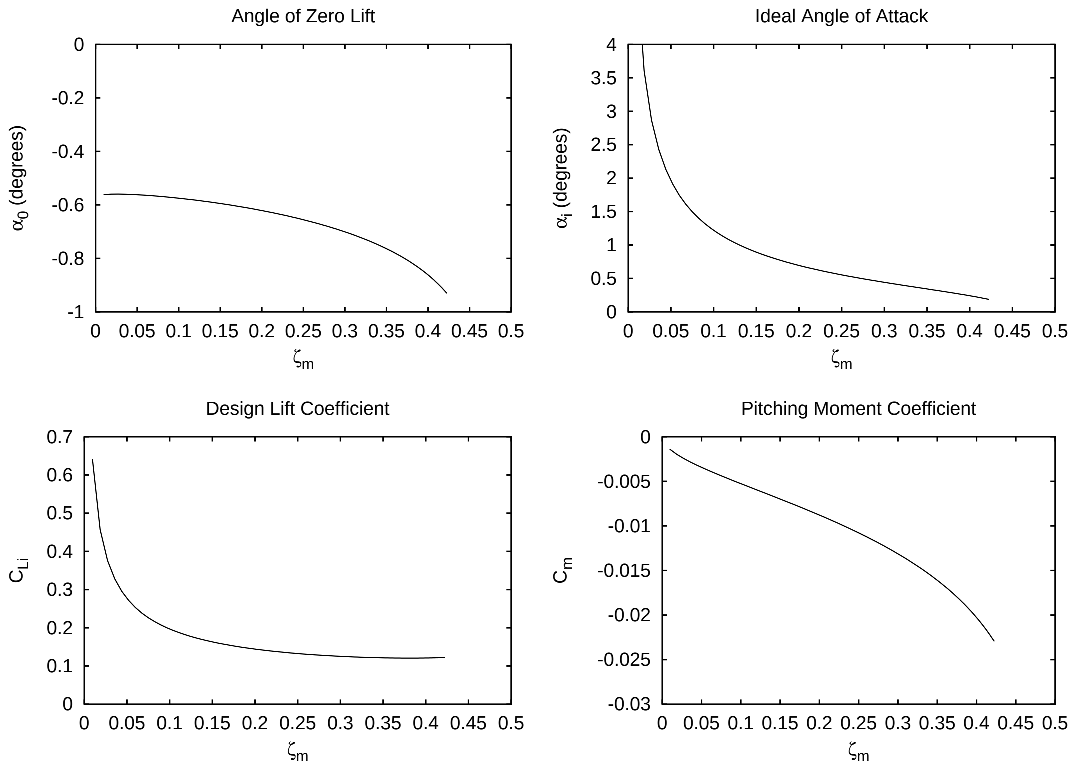

Figure 4: *The lift characteristics as a function of ζ_m for a NACA 5-digit airfoil with cambre $c = 0.01$.*

$$C_m = -\frac{k_1}{192}\Big[(32\zeta_j^3 - 16\zeta_j^2 - 28\zeta_j + 30)\sqrt{\zeta_j(1-\zeta_j)} + (24\zeta_j - 15)\arccos(1 - 2\zeta_j)\Big] \tag{47}$$

Figure 4 plots α_0, α_i, C_{Li} and C_m as a function of ζ_m for a NACA 5-digit airfoil with cambre equal to 1% of chord.

The class `NACA5DigitMeanLineOffset<F>` represents NACA 5-digit mean line offset curves. It is derived from the base class `PPSplineMeanLineOffset<F>` and has the following has the following public members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`NACA5DigitMeanLineOffset(F zetam, F c)`

Makes a NACA 5-digit mean line offset curve for an airfoil having cambre `c`. The maximum ordinate occurs when ζ is `zetam`. The value of `zetam` must not exceed $1 - 1/\sqrt{3} \approx 0.42265$.

`NACA5DigitMeanLineOffset(const Str &desig)`

Makes a NACA 5-digit mean line offset curve having designation `desig`. The string `desig` has three digits: three-halves the first gives the design lift coefficient in tenths; the second and third together give the location of the maximum mean line ordinate in twentieths of the chord. For example, the designation `"230"` indicates that $C_{Li} = 0.3$ and $\zeta_m = 0.15$.

`void set_cambre(F c)`

Redefines the mean line offset curve to have cambre `c`.

`void set_zeta_at_maximum_ordinate(F zetam)`

Sets the location of the maximum ordinate to `zetam`. The value of `zetam` must not exceed $1 - 1/\sqrt{3} \approx 0.42265$.

`F k1() const`

Returns the value of k_1.

`F zeta_at_join() const`

Returns the value of ζ_j: the point where the cubic and linear curve are joined.

6.1.1.4 Constant load mean line offset curve

According to thin airfoil theory, a mean line offset curve of the form

$$m(\zeta) = -\frac{C_{Li}}{4\pi}\Big[(1-\zeta)\ln(1-\zeta) + \zeta\ln\zeta\Big] \tag{48}$$

will produce a constant load distribution along the length of the airfoil when the angle of attack is zero. This type of mean line offset curve is used by the NACA 16-series airfoils. Since this mean line offset curve is symmetric about $\zeta = \frac{1}{2}$, its ideal angle of attack is zero, its angle of zero lift is $\alpha_0 = -C_{Li}/2\pi$ and its pitching moment coefficient is $C_m = -C_{Li}/4$. The maximum ordinate occurs when $\zeta = \frac{1}{2}$ and has the value $c = C_{Li}\ln 2/4\pi$.

Similarly, a mean line offset curve of the form

$$\begin{aligned} m(\zeta) = \frac{C_{Li}}{2\pi(a+1)}\Bigg\{&\frac{1}{4(1-a)}\Big[(a-\zeta)^2\ln\big((a-\zeta)^2\big) - (1-\zeta)^2\ln\big((1-\zeta)^2\big) \\ &+ (1-\zeta)^2 - (a-\zeta)^2\Big] - \zeta\ln\zeta + g - h\zeta\Bigg\} \end{aligned} \tag{49}$$

with

$$g = -\frac{1}{4}\left(a + 1 + \frac{2a^2\ln a}{1-a}\right); \quad h = \frac{(1-a)\big(2\ln(1-a) - 1\big)}{4} + g \tag{50}$$

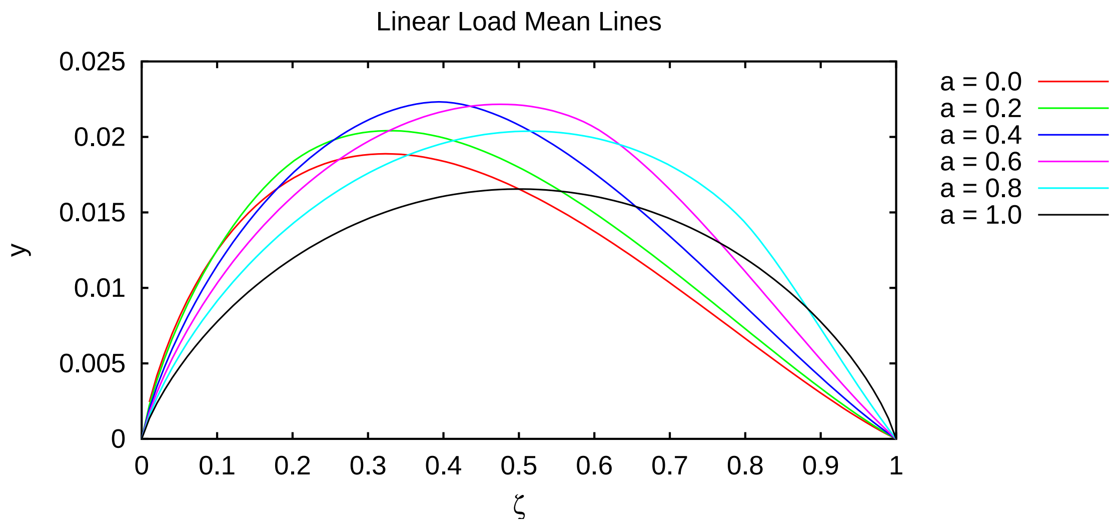

Figure 5: *Constant load mean line offset curves with $C_L = 0.3$ for six values of a.*

will generate a load distribution which is constant for $0 \leq \zeta \leq a$ and then decreases linearly to zero at the trailing edge; in the limit $a \to 1$, we get the constant load mean line of Equation (48). This type of mean line offset curve is used by the NACA 6-series airfoils. Its ideal angle of attack and angle of zero lift are

$$\alpha_i = \frac{C_{Li} h}{2\pi(a+1)}; \qquad \alpha_0 = \frac{C_{Li}}{2\pi}\left(\frac{h}{a+1} - 1\right) \tag{51}$$

Figure 5 plots the mean line offset curves of Equation (49) for six values of a; the curve with $a = 1$ is equivalent to the mean line of Equation (48).

The theoretical curves of Equations (48) and (49) have the problem that the slope is infinite at the leading edges and also at the trailing edge for the case of Equation (48); all higher derivatives are infinite at both leading and trailing edges. Abbot and von Doenhoff suggest alleviating this problem at the leading edge by generating the airfoil offsets using a circle of prescribed radius of curvature whose centre is on the line through the leading edge and has the slope of the mean line at $\zeta = 0.005$. In the same spirit, we avoid the problems at the leading and trailing edges by using a quadratic curve when $\zeta < 0.005$ or $\zeta > 0.995$. The quadratic coefficients are chosen to make the mean line offset curve continuous and smooth. For the case when $a = 0$ the deviation of the quadratic curves from the exact curves does not exceed 6.37×10^{-5} times the design lift coefficient, the maximum deviation occurring at about 0.1% of the chord. The deviation of the airfoil shape from the exact shape will be even smaller.

`ConstLoadMeanLineOffset` is a class which represents the mean line offset curves described by Equations (48) and (49). It is derived from the class `MeanLineOffset<F>`

and has the following members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`ConstLoadMeanLineOffset(F cl, F a)`
: Makes a mean line offset curve for an airfoil having lift coefficient `cl` with constant load for $0 \leq \zeta \leq a$. If `a` exceeds 1, $m(\zeta)$ will have the form given by Equation (48) with the adjustments at the leading and trailing edges described above. Otherwise $m(\zeta)$ will be of the form given by Equation (49) with similar adjustments at the leading and trailing edges. If `a` is negative it will be set to zero.

`void set_cambre(F c)`
: Redefines the mean line offset curve to have cambre `c`.

`F cambre() const`
: Returns the cambre.

`F design_lift_coefficient() const`
: Returns the design lift coefficient, C_{Li}.

`Angle<F> ideal_angle_of_attack() const`
: Returns the ideal angle of attack.

`F get_a() const`
: Returns the end of the region of constant loading.

6.1.1.5 Mean line offset curves which are circular arcs

An `ArcMeanLineOffset<F>` represents an airfoil mean line curve which is an arc of a circle.

$$m(\zeta) = -y + \sqrt{y^2 + \zeta(1-\zeta)}; \qquad y = \frac{1}{8c} - \frac{c}{2} \tag{52}$$

where c is the cambre. Because the mean line offset curve is symmetric about $\zeta = \frac{1}{2}$, we have

$$\alpha_i = 0; \qquad C_{Li} = -2\pi\alpha_0; \qquad C_m = \frac{\pi\alpha_0}{2} \tag{53}$$

The angle of zero lift, α_0, can be approximated using

$$\alpha_0 = -2c\big(1 + 2c^2 + 4c^4 + 12c^6 + 36c^8 + o(c^{10})\big) \tag{54}$$

`ArcMeanLineOffset<F>` has the following members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`ArcMeanLineOffset(F c)`
: Makes a circular arc mean line offset curve with cambre `c`.

`void set_cambre(F c)`
: Redefines the mean line offset curve to have cambre `c`.

6.1.1.6 Mean line offset curves derived from offsets

An `OffsetMeanLineOffset<F>` represents a mean line offset curve generated by splining a sequence of offsets: i.e. a sequence of (ζ, y) pairs. It is derived from the base class `PPSplineMeanLineOffset<F>`.

The angle of zero lift, ideal angle of attack, design lift coefficient and the pitching moment can all be calculated exactly (according to thin airfoil theory) for these mean line offset curves.

`OffsetMeanLineOffset<F>` has the following members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

```
typedef typename Spline::CubicSpline<F>::ValArray ValArray
```
The type of the sequence of offset values.

```
OffsetMeanLineOffset(const Spline::KnotSeq<F> &zeta_vals,
                     const ValArray &y_vals)
```
Makes a mean line offset curve for an airfoil by splining the offsets given by `zeta_vals` and `y_vals`. The lengths of `zeta_vals` and `y_vals` must be the same.

```
void define(const Spline::KnotSeq<F> &zeta_vals,
            const ValArray &y_vals)
```
Redefines the mean line offset curve by splining the offsets given by `zeta_vals` and `y_vals`. The lengths of `zeta_vals` and `y_vals` must be the same.

6.1.2 Classes to represent thickness curves

The class `ThicknessCurve<F>` represents an airfoil thickness curve. It is a curve, $t(\zeta)$, which returns the value of the half-thickness of the airfoil. The value of the thickness curve at $\zeta = 0$ must be zero. If $t(1) > 0$, the thickness curve is said to have an open trailing edge. If $t(1) = 0$ and $t'(1)$ is finite, the trailing edge is said to be sharp. If $t'(\zeta)$ approaches infinity as ζ approaches 1, the trailing edge is blunt. `ThicknessCurve<F>` allows three different methods for closing thickness curves which are open, two which make a sharp trailing edge and one which makes a blunt trailing edge.

`ThicknessCurve<F>` is derived from the base class `CurveLib::Curve<1U,F,F>` and has the following members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`ThicknessCurve(CurveLib::Curve<1U,F,F> crv, bool blunt = false)`

Makes an airfoil thickness curve using `crv`. The value `crv(0)` must equal zero; if not a `ProgError` is thrown. If `blunt` is true, the thickness curve is assumed to have a blunt trailing edge (it is not possible to determine this from `crv` alone); in this case `crv(1)` must equal zero; if not a `ProgError` is thrown.

`bool is_closed() const`

Returns `true` if the thickness curve is closed at the trailing edge: i.e. if its value at $\zeta = 1$ is zero.

`bool is_blunt() const`

Returns true if the thickness curve is blunt at the trailing edge: i.e. its derivative goes to infinity as one approaches the trailing edge. If the trailing edge is open, it is not blunt.

`void close_trailing_edge_by_extension()`

Ensures that the thickness curve has a closed trailing edge by extending the range of the thickness curve until it is zero. The parameter ζ is then scaled so that it is 1 at the new trailing edge location. The new thickness curve is defined by:

$$t_{new}(\zeta) = \begin{cases} t(\eta) & \text{for } \eta \leq 1 \\ t(1) - t'(1)(1-\eta) & \text{for } \eta \geq 1 \end{cases} \tag{55}$$

$$\eta = \zeta \left(1 - \frac{t(1)}{t'(1)}\right) \tag{56}$$

Due to the scaling of ζ, the thickness curve is modified throughout its length. The new trailing edge is sharp. This form of closure is illustrated in Figure 6.

If the trailing edge is already closed, the thickness curve is not changed.

An `Error` will be throw if the derivative of the thickness curve is not strictly negative at $\zeta = 1$.

`void close_trailing_edge_with_parabola(F zeta0 = F(1))`

Ensures that the thickness curve has a closed trailing edge by adding a parabolic curve to it starting at $\zeta = \zeta_0$; does nothing if the trailing edge is already closed.

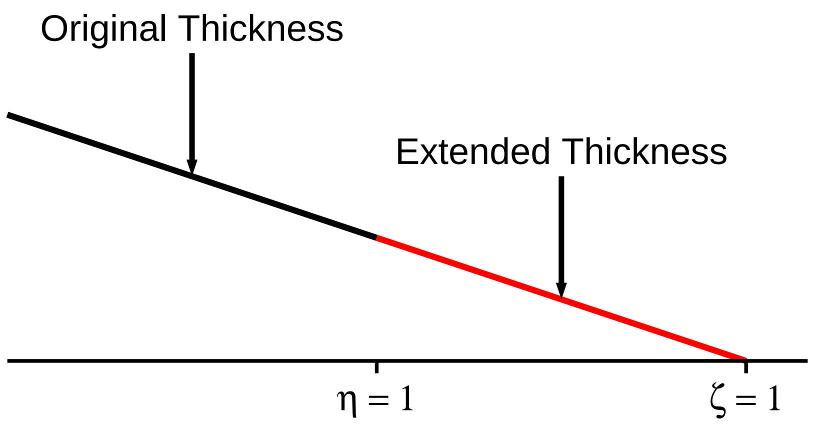

Figure 6: *Trailing edge closure by extension.*

The value of ζ_0 is given by `zeta0`. If it is outside the range [0,1), ζ_0 will be set to the location of maximum thickness.

The parabola is chosen so that the airfoil is smooth (i.e. the parabola has a derivative of zero at $\zeta = \zeta_0$).

$$t_{new}(\zeta) = \begin{cases} t(\zeta) & \text{for } \zeta \leq \zeta_0 \\ t(\zeta) - \dfrac{t(1)(\zeta - \zeta_0)^2}{(1 - \zeta_0)^2} & \text{for } \zeta \geq \zeta_0 \end{cases} \tag{57}$$

The new trailing edge is sharp. This form of closure is illustrated in Figure 7.

`void make_trailing_edge_blunt(F radius)`

If the thickness curve has an open trailing edge, closes it such that it is blunt; does nothing if the trailing edge is already closed. The approximate radius of curvature of the trailing edge is given by `radius`; it must be strictly positive.

The thickness curve is closed using a quarter elliptic arc with half width $t(1)$ and specified radius of curvature, r, at the trailing edge. The half-length of the ellipse is then $t(1)^2/r$. The new thickness curve is:

$$t_{new}(\zeta) = \begin{cases} t(\zeta) & \text{for } \zeta \leq \zeta_0 \\ t(\zeta) - t(1) + \dfrac{r\sqrt{(1 - 2\zeta_0 + \zeta)(1 - \zeta)}}{t(1)} & \text{for } \zeta \geq \zeta_0 \end{cases} \tag{58}$$

$$\zeta_0 = 1 - \frac{t(1)^2}{r} \tag{59}$$

The thickness curve is only modified in a small region close to the trailing edge. This form of closure is illustrated in Figure 8.

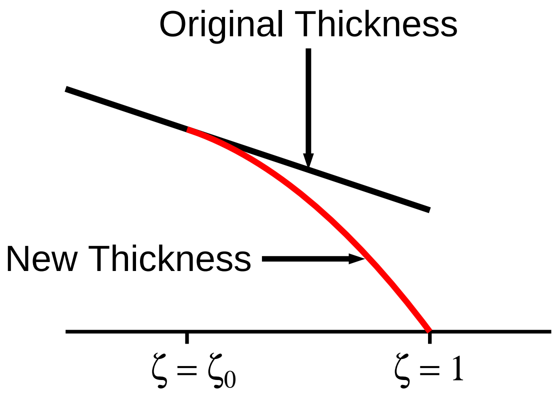

Figure 7: *Trailing edge closure with a parabola.*

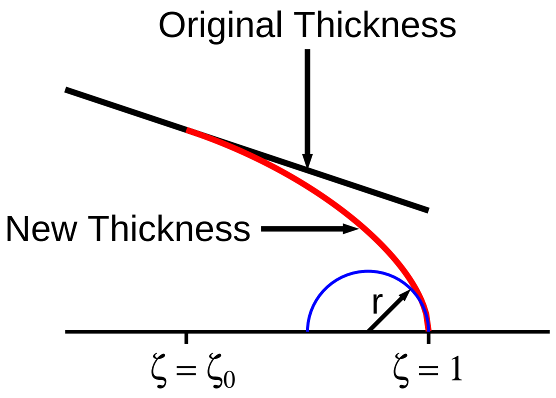

Figure 8: *Blunt trailing edge closure.*

6.1.2.1 NACA 4-digit thickness curves

The NACA 4-digit and 5-digit series airfoils use the following thickness curve:

$$t(\zeta) = \frac{t_{max}}{0.2}\left[0.29690\sqrt{\zeta} - 0.12600\zeta - 0.35160\zeta^2 + 0.28430\zeta^3 - 0.10150\zeta^4\right] \qquad (60)$$

where t_{max} is the maximum thickness. The class `NACA4DigitThicknessCurve<F>` represents this curve. It is derived from the class `ThicknessCurve<F>` and has the following members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`NACA4DigitThicknessCurve(F tmax)`
: Makes the thickness curve used by the NACA 4-digit and 5-digit series of airfoils. The maximum thickness will be `tmax`.

`NACA4DigitThicknessCurve(const Str &desig)`
: Makes the thickness curve used by the NACA 4-digit and 5-digit series of airfoils with thickness determined by the designation `desig`, a two-digit string giving the maximum thickness in percent of chord: for example, if `desig` is `"20"`, the maximum thickness will be 20% of the chord.

`void set_thickness(F tmax)`
: Changes the maximum thickness to `tmax`.

6.1.2.2 Thickness curve derived from offsets

An `OffsetThicknessCurve<F>` represents an airfoil thickness curve that is specified using a set of offsets and the radius of curvature at the leading edge. The offsets are given in terms of the chord fraction ζ and the thickness relative to the chord. The curve is of the form: $y = \sqrt{2r\zeta}$ + cubic spline. This ensures that the geometry at the leading edge is accurate. Thickness curves of this type are used by the NACA 6-series airfoils.

The class `OffsetThicknessCurve<F>` represents thickness curves specified using offsets and a leading edge curvature. It is derived from `ThicknessCurve<1U,F,F>` and has the following constructor as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`OffsetThicknessCurve(F r, unsigned n, const F *zeta, const F *yt)`

Makes an airfoil thickness curve having leading edge radius `r` (relative to chord) and `n` offsets in `zeta` and `yt`. The values in `zeta` must be positive and strictly increasing. The last value of zeta must be 1.0. The values in `yt` must be positive except that the value corresponding to $\zeta = 0$ must be zero and the value corresponding to $\zeta = 1$ may be 0.

6.1.3 NACA airfoils

The National Advisory Committee for Aeronautics (NACA) has designed several series of airfoils that have been used for decades in many applications. The Airfoil Library currently implements NACA 4-digit, 5-digit, 16-series and 6-series airfoils [2]. They may all be represented using the class `NACAAirfoil<F>`. Since the NACA airfoils are all defined using a thickness curve and a mean line offset curve, `NACAAirfoil<F>` is derived from the class `ThickDistAirfoil<F>`. All NACA airfoils are canonical.

`NACAAirfoil<F>` has the following members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`NACAAirfoil(const Str &desig)`

Creates a NACA airfoil with designation `desig`. If the designation is not known, an `UnknownNACADesignation` exception is thrown (it is derived from `Error`). The following designations are understood where `d` stands for any digit:

`dddd`: a 4-digit series airfoil. The first digit gives the cambre in percent of the chord, the second gives the location of the maximum ordinate in tenths of the chord and the last two give the maximum thickness in percent of chord. For example, the designation `2415` indicates $c = 0.02$, $\zeta_m = 0.4$ and $t = 0.15$.

`ddddd`: a 5-digit series airfoil. Three-halves the first digit gives the design lift coefficient in tenths; the second and third digits together give the location of the maximum mean line ordinate in twentieths of the chord; the last two digits give the maximum thickness in percent of chord. For example, the designation `"23012"` indicates that $C_{Li} = 0.3$, $\zeta_m = 0.15$ and $t = 0.12$.

`16-ddd`: a 16-series airfoil. The first digit following the hyphen gives the design lift coefficient in tenths, and the last two digits give the maximum thickness in percent of chord. For example, the designation `"16-212"` indicates that $C_{Li} = 0.2$ and $t = 0.12$.

`6n_d-ddd a = d.d`: a 6-series airfoil. Here `n` is one of the digits 3, 4, 5 or 6; it gives the location of the in tenths of chord downstream of the leading edge of the minimum pressure for the zero cambre airfoil at zero lift.

The underscore denotes a subscript; thus, for example, `63_3-018` corresponds to the airfoil 63_3–018. The subscripted digit gives the range of lift coefficient in tenths above and below the design lift coefficient in which favourable pressure gradients exist on both sides of the airfoil. It is omitted if the thickness is less than 12% of the chord length.

The first digit after the hyphen and the characters beginning with `a` serve to specify the mean line offset curve which is of the constant load type (see Section 6.1.1.4). The digit after the hyphen gives the design lift coefficient in tenths, while the characters beginning with `a` specify the value of a in Equation (49); the spaces around the equals sign are optional. If the characters beginning with `a` are omitted, Equation (48) will be used to define the mean line offset curve.

The thickness curves of the 6-series airfoils are not derived from simple formulae as are the 4 and 5-digit airfoils. Nor can one thickness curve be derived from another simply by scaling the curves with maximum thickness. Instead, the thickness curves are obtained from data tabulated by Abbot and von Doenhoff [2]. The following airfoils can be represented (the string in brackets is the corresponding value of `desig`):

6n–d06 (`6n-d06`)
6n–d08 (`6n-d08`)
6n–d09 (`6n-d09`)
6n–d10 (`6n-d10`)
$6n_1$–d12 (`6n_1-d12`)
$6n_2$–d15 (`6n_2-d15`)
$6n_3$–d18 (`6n_3-d18`)
$6n_4$–d21 (`6n_4-d21`)

An exception is that there is no representation for 63–d08. The last two digits give the maximum thickness in percent of chord.

For example, if `desig` is `"66_4-221 a = 0.8"`, then the NACA designation of the airfoil is 66_4–221 $a = 0.5$. It has a thickness of 0.21 and mean line offset curve defined by Equation (49) with $C_{Li} = 0.2$ and $a = 0.5$.

`6n_(ddd)-ddd a = d.d`: a 6-series airfoil. As with the 6-series designation of the form `6n_d-ddd a = d.d`, the digit `n` can be one of 3, 4, 5 or 6, the underscore denotes a subscript and the first digit after the hyphen and the characters beginning with `a` specify the mean line. The last two digits give the maximum thickness in percent of chord. The thickness distribution is scaled from the thickness distribution specified by the digits in brackets. Let these digits be `(pqr)`. Then the thickness curve is obtained by scaling the thickness curve of the airfoil `6n_p-0qr`. If there are only two digits in brackets, it is assumed that `p` is missing and the thickness curve to be scaled is obtained from the airfoil `6n-0qr`. From the restrictions on the airfoils with designation of the form `6n_d-ddd`, the digits in brackets must be one of 06, 08, 09, 10, 112, 215, 318 or 421 except that 08 is not allowed if `n` is 3.

For example, if `desig` is `"65_(318)-217 a = 0.5"`, then the NACA designation of the airfoil is $65_{(318)}$–217 $a = 0.5$. This airfoil has a thickness curve obtained from the airfoil with designation 65_3–018 but scaled so that its maximum thickness is 0.17.

`void define(const Str &desig)`
Redefines the airfoil to be a NACA airfoil with designation `desig`. If the designation is not known, an `UnknownNACADesignation` exception is thrown.

`void define_4_digit(F t, F zetam, F c)`
Defines the airfoil to be a member of the 4-digit NACA series having thickness `t` and cambre `c`. The maximum mean line ordinate occurs at ζ equal to `zetam`.

A `NACA4DigitThicknessCurve<F>` is used to represent the thickness curve: see Section 6.1.2.1. A `NACA4DigitMeanLineOffset<F>` is used to represent the mean line offset curve: see Section 6.1.1.2.

`void define_5_digit(F t, F zetam, F c)`
Defines the airfoil to be a member of the 5-digit NACA series having thickness `t` and cambre `c`. The maximum mean line ordinate occurs at ζ equal to `zetam`.

The thickness curve is the same as for the 4-digit series (see Section 6.1.2.1). The mean line offset curve is represented by a `NACA5DigitMeanLineOffset<F>`: see Section 6.1.1.3.

`void define_16_series(F t, F cl)`
Defines the airfoil to be a member of the NACA 16-series having thickness `t` (fraction of chord) and design lift coefficient `cl`.

The thickness curves for the 16-series airfoils are directly proportional to the thickness. The curve used here is derived by splining the offsets given by Abbot and von Doenhoff for the 16-series airfoil with thickness of 21% of chord, then scaling it according to the value of `t`.

The mean line offset curve for a 16-series airfoil is of the constant load type (Equation (48)) and is represented by a `ConstLoadMeanLineOffset<F>`.

`void define_6_series(F t, F xpmin, F low_drag, F cl, F a)`

Defines the airfoil to be a member of the NACA 6-series having thickness `t` (fraction of chord), low_drag range `low_drag`, minimum pressure at `xminp` (fraction of chord), design lift coefficient `cl`, and constant loading for ζ less than `a`.

The thickness curves for the 6-series airfoils are calculated by splining the offsets given by Abbot and von Doenhoff. This limits the values of `xpmin` to 0.3, 0.4, 0.5 and 0.6, and `low_drag` and `t` to the following pairs: $(\texttt{low\_drag}, \texttt{t}) = (0, 0.06)$, (0,0.08), (0,0.09), (0,0.1), (0.1,0.12), (0.2,0.15), (0.3,0.18), (0.4,0.21) with the exception that (0,0.08) is not allowed when `xpmin` is 0.3.

The mean line offset curve for a 6-series airfoil is of the constant load type with a linear drop off to the trailing edge (Equation (49)) and is represented by a `ConstLoadMeanLineOffset<F>`.

The ideal angle of attack, the angle of zero lift and the design lift coefficient are calculated exactly for all NACA airfoils. The pitching moment about one quarter chord is calculated exactly for 4-digit, 5-digit and 16-series airfoils; for the 6-series airfoils it will be approximate.

6.1.4 The DTMB modification of the NACA 66 airfoil

An airfoil that has often been used for propeller sections is a modification of the NACA 66 airfoil designed at the David Taylor Model Basin (DTMB). The modifications thicken the trailing edge for ease of manufacture and fair the nose to eliminate a bump in the pressure distribution that occurs on the NACA 66 airfoils. In addition, airfoils of different thicknesses are obtained simply by scaling the airfoil with thickness equal to 6% of chord (the NACA 66 airfoils have slightly different thickness curves for each thickness). A constant load mean line offset curve with $a = 0.8$ is used. The complete geometry of the airfoil is described by Brockett [5].

The class `NACA66DTMBmod<F>` implements the DTMB modification of the NACA 66 airfoil. It is derived from the base class `ThickDistAirfoil<F>` and has the following members as well as the default and copy constructors, virtual destructor, assignment operator and members inherited from its base classes.

`NACA66DTMBmodAirfoil(F t, F cl)`

Makes an airfoil having thickness `t` (relative to chord length) and design lift coefficient `cl`.

`void define(F t, F cl)`
: Redefines the airfoil to have thickness `t` (relative to chord length) and design lift coefficient `cl`.

`void set_cambre(F c)`
: Redefines the airfoil to have cambre `c` (relative to chord length).

`F cambre() const`
: Returns the cambre of the airfoil.

`F design_lift_coefficient() const`
: Returns the design lift coefficient for the airfoil.

`Angle<F> ideal_angle_of_attack() const`
: Returns the ideal angle of attack for the airfoil. The class `Angle<F>` is described in Reference 1, Annex E.

7 Defining airfoils from files

Airfoils can be defined by reading records from a file in OFFSRF format [6]. The class `AirfoilReader<F>` provides the interface between the file and the airfoil created. Its template argument, `F`, is the type of the floating point numbers used for the airfoil; it will normally be `float` or `double`.

For example, to define an airfoil from data in the file `afoil.dat`, one could use the following code:

```
// Open an OFFSRF input stream
Offsrf::IFStream in("afoil.dat");
if (!in) throw Error("Could not open file afoil.dat");

// Read the file
AirfoilReader<double> reader;
in >> reader;
if (!in) throw Error("Error in file afoil.dat");

// Get the airfoil
Airfoil<F> afoil = reader.airfoil;
if (!afoil.is_defined()) {
  throw Error("No airfoil was specified in the file afoil.dat");
}
```

The class `Offsrf::IFStream` is simply a `std::istream` adapted for use by the OFFSRF classes.

`AirfoilReader<F>` also allows a name to be given to the airfoil. The name can be retrieved from its public member `name`:

```
std::string aname = AirfoilReader<F>::name;
```

`AirfoilReader<F>` recognizes records with the following names: `CLOSE TRAILING EDGE`, `COMMENT`, `INCLUDE`, `JOUKOWSKI`, `NACA`, `NACA 66 DTMB(mod)`, `NAME`, `OFFSETS`, and `THICKNESS DISTRIBUTION AIRFOIL`. The `COMMENT` record is a standard OFFSRF record which simply writes its contents to `stdout`. The `INCLUDE` record is also a standard OFFSRF record. It directs input to another file; the content of the record is interpreted as the name of the file: see Reference 6. The remaining records are described in the following sections. Each record is optional but there must be one of `JOUKOWSKI`, `NACA`, `OFFSETS` or `THICKNESS DISTRIBUTION AIRFOIL` or else the airfoil will remain undefined.

7.1 The CLOSE TRAILING EDGE record

The `CLOSE TRAILING EDGE` record has no data: i.e. it simply takes the form:

```
{CLOSE TRAILING EDGE}
```

When it is present, the airfoil will be closed. If the trailing edge is already closed, the presence of this record will cause no change. The trailing edge is closed using the function `Airfoil<double>::close_trailing_edge()` (see Section 4) unless a `BLUNT TRAILING EDGE` record is also present, in which case the function `Airfoil<double>::make_trailing_edge_blunt(F radius)` is used.

7.2 BLUNT TRAILING EDGE

Presence of a `BLUNT TRAILING EDGE` causes the trailing edge to be closed using the function `Airfoil<double>::make_trailing_edge_blunt(F radius)`. If the trailing edge is already closed, this record will cause no change. The record has the form

```
{BLUNT TRAILING EDGE: $\alpha$ }
```

where the value α is used to specify the radius of curvature at the trailing edge. Let the distance separating the points at the trailing edge be Δ. Then the radius of curvature will be $\Delta/2\alpha$ and the change in the airfoil curve extends roughly $\frac{1}{2}\alpha\Delta$ upstream of the trailing edge. If $\alpha = 1$, the trailing edge closure will roughly be a semi-circle; larger values of α make the trailing edge sharper, smaller values more blunt.

7.3 The JOUKOWSKI record

This record defines a Joukowski airfoil. It has the following format:

```
{JOUKOWSKI: thickness cambre
  {FINITE TE DERIVATIVE} ! optional
}
```

The value of the cambre can be omitted; in that case the cambre will be zero.

If the `FINITE TE DERIVATIVE` record is present, then the ξ-parameterization defined by Equation (17) is used to ensure that the first derivative of the airfoil ξ-curve will be finite at the trailing edge; otherwise the parameterization defined by Equation (16) is used and the first derivative at the trailing edge will be zero.

The airfoil is given the name `Joukowski t=`*thickness* `c=`*cambre*. For example, the record

```
{JOUKOWSKI: 0.06 0.02 }
```

results in an airfoil with the name `Joukowski t=0.06 c=0.02`.

The airfoil and its name replace any airfoil or name created by records occurring earlier in the file.

7.4 The NACA record

This record defines a NACA airfoil from a designation. It has the following format:

```
{NACA: designation }
```

where *designation* is any of the designations recognized by the `NACAAirfoil<F>` constructor: see Section 6.1.3. Leading and trailing whitespace is stripped from the designation before being passed to the constructor.

The airfoil is given the name `NACA` *designation*. For example, the record

```
{NACA: 63_3-418 a=0.7 }
```

results in an airfoil with the name `NACA 63_3-418 a=0.7`.

The airfoil and its name replace any airfoil or name created by records occurring earlier in the file.

7.5 The NACA 66 DTMB(mod) record

This record defines a DTMB modification of the NACA 66 airfoil from specified values for the thickness and cambre. It has the following format:

`{NACA 66 DTMB(mod):` *thickness cambre* `}`

The value of the cambre can be omitted in which case it will be set to zero.

The airfoil is given the name `NACA 666 DTMB(mod) t=`*thickness* `c=`*cambre*. For example, the record

```
{NACA 66 DTMB(mod): 0.06 0.02 }
```

results in an airfoil with the name `NACA 66 DTMB(mod) t=0.06 c=0.02`.

The airfoil and its name replace any airfoil or name created by records occurring earlier in the file.

7.6 The NAME record

The `NAME` record is used to specify a name for the airfoil. It has the following format:

`{NAME:` *name* `}`

Leading and trailing whitespace will be stripped from the name. The name replaces any name created by records occurring earlier in the file.

7.7 The OFFSETS record

This record defines an airfoil from a set of (x, y) offsets. It has the following format:

```
{OFFSETS
  x0 y0
  ...
  xn yn
  {LEADING EDGE INDEX: ile } ! optional
  {BLUNT TRAILING EDGE} ! optional
}OFFSETS
```

The offsets must be ordered with increasing ξ: i.e. they start at the trailing edge, proceed to the leading edge along the pressure side, then back to the trailing edge along the suction side.

One of the offset points must lie at the leading edge. The index *ile* marks which point this is (note that the numbering starts at zero). If this record is not present, it is assumed that the leading edge is at $(0, 0)$; the offset point with that value is found and the value of *ile* set accordingly; if no offset with that value is present, an `Error` is thrown.

If the `BLUNT TRAILING EDGE` record is present, the trailing edge will be made blunt. The method for doing this depends on whether the first and last offset points are the same. If they are the same, the offsets are splined using a periodic cubic spline (see Reference 4, Section 8.1.3); the trailing edge curvature cannot be controlled. If the first and last offset points differ, an ordinary cubic spline is used and then the airfoil is closed using the algorithm described in Annex B.

Note that a `BLUNT TRAILING EDGE` record can be placed either inside or outside the `OFFSETS` record. If the first and last offset points differ, the resulting airfoil will be the same. However, if the first and last offset points are the same, the two airfoils will differ. When it is outside the `OFFSETS` record, the airfoil surface will be generated using an ordinary spline; since the resulting airfoil is closed, the `BLUNT TRAILING EDGE` will do nothing. However, when it is inside the `OFFSETS` record, a periodic spline is used and the airfoil trailing edge will be blunt. This behaviour is illustrated in Figure 9.

The airfoil replaces any airfoil created by records occurring earlier in the file.

7.8 The THICKNESS DISTRIBUTION AIRFOIL record

The `THICKNESS DISTRIBUTION AIRFOIL` record defines a thickness distribution airfoil (a `ThickDistAirfoil<F>`). It consists solely of sub-records whose names can be any of the records recognized by `ThickDistAirfoilreader<F>` (see Section 7.9):

`{THICKNESS DISTRIBUTION AIRFOIL`
  Any record recognized by `ThickDistAirfoilReader<F>`
`}THICKNESS DISTRIBUTION AIRFOIL`

The airfoil replaces any airfoil created by records occurring earlier in the file.

7.9 Defining thickness distribution airfoils from files

Because of the importance of thickness distribution airfoils, a special reader class is available for defining a `ThickDistAirfoil<F>` (see Section 6.1) from a file. It can be used in exactly the same way as `AirfoilReader<F>` but in place of an `Airfoil<F>` it contains a `ThickDistAirfoil<F>`. For example, to define a thickness distribution airfoil from data in the file `afoil.dat`, one could use the following code:

```
{OFFSETS
  1.00 0.0
  0.75 -0.07
  0.50 -0.1
  0.25 -0.1
  0.10 -0.07
  0.00 0.0
  0.10 0.07
  0.25 0.1
  0.50 0.1
  0.75 0.07
  1.00 0.0
  {BLUNT TRAILING EDGE}
}OFFSETS
```

```
{OFFSETS
  1.00 0.0
  0.75 -0.07
  0.50 -0.1
  0.25 -0.1
  0.10 -0.07
  0.00 0.0
  0.10 0.07
  0.25 0.1
  0.50 0.1
  0.75 0.07
  1.00 0.0
}OFFSETS
{BLUNT TRAILING EDGE}
```

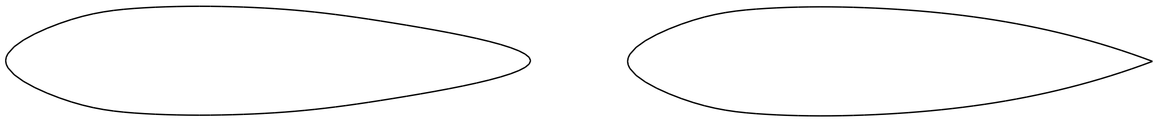

Figure 9: *The difference between airfoils generated with a* ***`BLUNT TRAILING EDGE`*** *record inside (right) and outside (left) an* ***`OFFSETS`*** *record. The airfoil on the left is generated using a periodic spline. The airfoil on the right is generated using an ordinary spline; since it is closed, the* ***`BLUNT TRAILING EDGE`*** *record does nothing in this case.*

```
// Open an OFFSRF input stream
Offsrf::IFStream in("afoil.dat");
if (!in) throw Error("Could not open file afoil.dat");

// Read the file
ThickDistAirfoilReader<double> reader;
in >> reader;
if (!in) throw Error("Error in file afoil.dat");

// Get the airfoil
ThickDistAirfoil<F> afoil = reader.airfoil;
if (!afoil.is_defined()) {
  throw Error("No airfoil was specified in the file afoil.dat");
}
```

One can now use the member functions specific to `ThickDistAirfoil<F>` which would not be available if an `AirfoilReader<F>` had been used.

In addition to the standard OFFSRF records `COMMENT` and `INCLUDE` (see Reference 6), `ThickDistAirfoilReader<F>` recognizes the records NACA, NACA 66 DTMB(mod) and NAME and treats them exactly as does `AirfoilReader<F>`. It also allows the airfoil to be specified from a mean line and thickness curve using any records recognized by the classes `MeanLineOffsetReader<F>` and `ThicknessCurveReader<F>` described in Sections 7.9.1 and 7.9.2.

7.9.1 A class for defining mean line offset curves from a file

The class `MeanLineOffsetReader<F>` can be used to define mean line offset curves from a file in OFFSRF format. In addition to the standard OFFSRF records `COMMENT` and `INCLUDE` (see Reference 6), `MeanLineOffsetReader<F>` recognizes records with the following format:

`{CONSTANT LOAD MEAN LINE:` C_{Li} a `}`
`{MEAN LINE OFFSETS`
  ζ_0 y_0
  $\cdots$
  ζ_n y_n
`}MEAN LINE OFFSETS`
`{NACA 4-DIGIT MEAN LINE:` c ζ_m `}`
`{NACA 5-DIGIT MEAN LINE:` c ζ_m `}`
`{CIRCULAR ARC MEAN LINE:` c `}`

The parameters C_{Li} and a in the `CONSTANT LOAD MEAN LINE` record are used to define a constant load mean line offset curve according to Equation (49). If a is not present, the mean line offset curve defined by Equation (48) is used.

The parameters c and ζ_m are the cambre and the location of the maximum mean line offset.

Only one of these records is necessary; if more than one is present, the last one is used.

7.9.2 A class for defining thickness curves from a file

The class `ThickCurveReader<F>` can be used to define thickness curves from a file in OFFSRF format. In addition to the standard OFFSRF records `COMMENT` and `INCLUDE` (see Reference 6), `ThickCurveReader<F>` recognizes records with the following format:

`{NACA 4-DIGIT THICKNESS:` *thickness* `}`
`{THICKNESS OFFSETS`
  $\zeta_0\ y_0$
  `...`
  $\zeta_n\ y_n$
`}THICKNESS OFFSETS`
`{BLUNT TRAILING EDGE:` α `}`
`{CLOSE TRAILING EDGE:` *method* `}`

The value α in the `BLUNT TRAILING EDGE` record is used to specify the radius of curvature at the trailing edge. The radius of curvature will be $t(1)/\alpha$ where $t(1)$ is the value of the thickness curve at the trailing edge. The change in the airfoil curve extends roughly $\alpha t(1)$ upstream of the trailing edge. If $\alpha = 1$, the trailing edge closure will roughly be a semi-circle.

The parameter *method* in the `CLOSE TRAILING EDGE` is optional. If present it should be one of `extension` or `parabola`. If *method* is not present, the trailing edge is closed using the default method (a parabola starting at the maximum thickness). If *method* is `parabola` the ζ-value at which the parabola starts can be specified:

`{CLOSE TRAILING EDGE: parabola` ζ `}`

If the value ζ is not present the location of maximum thickness is used.

Only one of `CLOSE TRAILING EDGE` or `BLUNT TRAILING EDGE` is necessary; if more than one is present, the last one is used. Similarly, only one of the records `NACA 4-DIGIT THICKNESS` or `THICKNESS OFFSETS` is necessary.

8 Defining an airfoil from the command line

For programs requiring the definition of a single airfoil, it is convenient to be able to specify the airfoil using command line arguments. The function `make_airfoil` provides a means for doing this.

`Airfoil<double> make_airfoil(const std::vector<Str> &args, Str &name)`

Parses the command line arguments given by `args`, then makes an airfoil. The type `Str` is an alias for `std::string`. The first element of `args` is the program name. The following command line arguments are recognized. Arguments in square brackets are optional.

file

Specifies a file which will be read using an `AirfoilReader<double>`.

`-D` *t c*

Makes a DTMB modified NACA 66 airfoil with thickness t and cambre c.

`-j` *t c* `[` *te-flag* `]`

Makes a Joukowski airfoil with thickness t and cambre c. If *te-flag* is 1, the first derivative of the airfoil ξ-curve will be finite at the trailing edge. Otherwise it will be zero. The default is *te-flag* $= 0$.

`-NACA` *desig*

Defines the airfoil from the NACA designation *desig*. See Section 6.1.3 for a description of the format of *desig*.

`-cte [` *method* `[` *arg* `] ]`

Closes the trailing edge if it is not already closed. The optional argument *method* can be one of `p` (close using a parabola: see Section 6.1), `e` (close by extension: see Section 6.1) or `b` (blunt closure). The methods `e` and `p` should only be used in conjunction with the `-NACA` or `-D` options. If *method* is `p`, the optional *arg* gives the value ζ_0 at which the parabola begins; if it is missing the parabola will start at the location of maximum offset of the mean line. If *method* is `b`, the optional *arg* gives the value of α for the closure where $\alpha = \Delta/2r$ with Δ equal to the separation of the points at the trailing edge and r equal to the radius of curvature at the trailing edge. If *arg* is missing, α is set to 1. If *method* is missing, the default trailing edge closure is used. If the trailing edge is already closed, this option does nothing.

If a name for the airfoil can be determined it is returned in `name`.

The following function is also provided for generating descriptions of the allowed command line arguments.

`void make_airfoil_usage(FormattedOStream &out)`

Writes information on the allowed command line arguments for `make_airfoil` to `out`. A `FormattedOStream` is an adapted output stream which allows the output to be indented and automatically wrapped at a specified column. It is described in more detail in Annex C.

Here is an example of the use of `make_airfoil` in a program that also defines command line options `-h` and `-x`. The option `-h` writes a message describing the usage of the program to `stdout`, then exits the program.

```
#include "MakeAirfoil.h"
#include "Airfoil.h"
#include "FormattedOStream.h"

void usage()
```

```
{
  FormattedOStream out(std::cout);
  out << "Usage: my_program\n";
  out.rel_indent(7);
  out << "[ -x ]          ! Description of -x\n";
  out << "[ -h ]          ! Write this help message\n";
  make_airfoil_usage(out);
}

int main(int argc, char **argv)
{
  try {
    // Parse the command line arguments
    std::vector<Str> afoil_args;
    afoil_args.push_back(argv[0]);

    int arg = 1;
    while (arg < argc) {
      if (Str(argv[arg]) == "-x") {
        arg++;
        // Handle option -x
      }
      else if (Str(argv[arg]) == "-h") {
        usage();
        return 1;
      }
      else {
        afoil_args.push_back(argv[arg++]);
      }
    }

    // Make the airfoil
    Str name;
    Airfoil<double> afoil = make_airfoil(afoil_args,name);

    // The rest of the program
    . . .

    return 0;
  }
  catch(Error &e) {
    std::cerr <{}< e.get_msg() <{}< ’\n’;
    return 1;
  }
}
```

If invoked with

```
my_program -h
```

the following output will be written to `stdout`.

```
Usage: my_program
       [ -x ]          ! Description of -x
       [ -h ]          ! Write this help message
       [ file ]        ! A file defining the airfoil in OFFSRF format
       [ -D t c ]      ! Makes a DTMB modified NACA 66 airfoil with
                       !   thickness t and cambre c.
       [ -NACA desig ] ! Defines the airfoil from a NACA designation
       [ -j t c [ te_flag ] ]
                       ! Makes a Joukowski airfoil with thickness t
                       !   and cambre c.  If te_flag is 1, the first
                       !   derivative of the airfoil xi-curve will be
                       !   finite at the trailing edge.  Otherwise it
                       !   will be zero (the default).
       [ -cte [ method [ arg ] ]]
                       ! Close trailing edge.  method can be e (close
                       !   by extension), p (close with parabola) or
                       !   b (blunt closure).  e and p can only be
                       !   used in conjunction with -NACA or -D.  If
                       !   method is p, then the optional arg gives
                       !   the starting zeta value for the parabola.
                       !   If method is b, then the optional arg gives
                       !   the value of alpha; the airfoil is modified
                       !   over a distance alpha times the trailing
                       !   edge gap.  If method is not present, a
                       !   default closure is used.
```

9 Concluding remarks

The Airfoil library is a collection of C++ classes for representing the geometry of airfoils. It is based on the CurveLib library [1] for representing differentiable curves. Several families of airfoils are represented explicitly in the Airfoil library: Joukowski airfoils, NACA 4-digit, 5-digit, 16-series and 6-series airfoils [2], and the David Taylor Model Basin (DTMB) modification of the NACA 66 airfoil commonly used in marine propellers. Generic airfoils can be generated from offsets on the airfoil surface, or from mean lines and thickness curves. Methods for closing the trailing edges of airfoils with trailing edge gaps are also implemented (Section 6.1.2 and Annex B).

Classes are also provided for defining airfoils from data in a file in OFFSRF format [6] and for specifying an airfoil from the program command line.

References

[1] Hally, D. (2006), C++ classes for representing curves and surfaces: Part I: Multi-parameter differentiable functions, (DRDC Atlantic TM 2006-254) Defence R&D Canada – Atlantic.

[2] Abbott, I. H. and von Doenhoff, A. E. (1958), Theory of Wing Sections, Dover Publications Inc., New York.

[3] Fritsch, F. N. and Carlson, R. E. (1980), Monotone Piecewise Cubic Interpolation, *SIAM Journal of Numerical Analysis*, 17, 238.

[4] Hally, D. (2006), C++ classes for representing curves and surfaces: Part II: Splines, (DRDC Atlantic TM 2006-255) Defence R&D Canada – Atlantic.

[5] Brockett, T. (1966), Minimum Pressure Envelopes for Modified NACA-66 Sections with NACA a = 0.8 Camber and BuShips Type I and Type II Sections, (Report 1780) David W. Taylor Naval Ship Research and Development Center.

[6] Hally, D. (1994), C++ Classes for Reading and Writing files in OFFSRF Format, (DREA Tech.Comm. 94/302) Defence R&D Canada – Atlantic.

[7] Hally, D. (2006), C++ classes for representing curves and surfaces: Part IV: Distribution functions, (DRDC Atlantic TM 2006-257) Defence R&D Canada – Atlantic.

This page intentionally left blank.

Annex A: Integrating mean lines

Thin airfoil theory provides expressions for the angle of zero lift, the ideal angle of attack and the pitching moment coefficient in terms of integrals of a canonical mean line along the chord line: see Equations (32)–(34). This annex discusses the methods for evaluating these integrals.

First suppose that the mean line is a polynomial,

$$p(\zeta) = \sum_{n=0}^{N} c_n \zeta^n \tag{A.1}$$

and we wish to evaluate the portion of the integral on $\zeta \in [a, b]$. Then it is clearly sufficient to be able to evaluate the integrals

$$J_{i,n}(\zeta) = \int \zeta^n f_i(\zeta)\, d\zeta \tag{A.2}$$

for $i = 1$, 2 or 3. Let

$$I_n(\zeta) = \int \frac{\zeta^{n-1/2}}{(1-\zeta)^{3/2}}\, d\zeta \tag{A.3}$$

Integrate by parts to get

$$\begin{aligned} I_n(\zeta) &= \zeta^n I_0(\zeta) - n \int \zeta^{n-1} I_0(\zeta)\, d\zeta && \text{(A.4)} \\ &= \zeta^n I_0(\zeta) - 2n \int \frac{\zeta^{n-1/2}}{(1-\zeta)^{1/2}}\, d\zeta && \text{(A.5)} \\ &= \zeta^n I_0(\zeta) - 2n\big(I_n(\zeta) - I_{n+1}(\zeta)\big) && \text{(A.6)} \end{aligned}$$

Therefore

$$I_{n+1}(\zeta) = \frac{(2n+1)I_n(\zeta) - \zeta^n I_0(\zeta)}{2n} \tag{A.7}$$

provided that $n > 0$. The first few $I_n(\zeta)$ are

$$\begin{aligned} I_{-1}(\zeta) &= \frac{2(2\zeta - 1)}{\sqrt{\zeta(1-\zeta)}} && \text{(A.8)} \\ I_0(\zeta) &= 2\sqrt{\frac{\zeta}{1-\zeta}} && \text{(A.9)} \\ I_1(\zeta) &= \arcsin(1-2\zeta) + I_0(\zeta) && \text{(A.10)} \\ I_2(\zeta) &= \frac{3\arcsin(1-2\zeta) + (3-\zeta)I_0(\zeta)}{2} && \text{(A.11)} \\ I_3(\zeta) &= \frac{15\arcsin(1-2\zeta) + (15 - 5\zeta - 2\zeta^2)I_0(\zeta)}{8} && \text{(A.12)} \end{aligned}$$

Now we can write

$$J_{1,n}(\zeta) = I_n(\zeta) \tag{A.13}$$

$$J_{2,n}(\zeta) = I_n(\zeta) - 3I_{n+1}(\zeta) + 2I_{n+2}(\zeta) \tag{A.14}$$

$$J_{3,n}(\zeta) = I_{n-1}(\zeta) - 2I_n(\zeta) \tag{A.15}$$

The integrals of the polynomial over $[a, b]$ are then:

$$\int_a^b p(\zeta) f_i(\zeta)\, d\zeta = \sum_{n=0}^{N} c_n \big(J_{i,n}(b) - J_{i,n}(a)\big) \tag{A.16}$$

for $i = 1$, 2 or 3. Notice that if $a = 0$, then $c_0 = 0$, since the mean line is canonical. We can then increase the lower limit of n to 1. Since $I_n(0)$ is well-defined for all $n \geq 0$, $J_{i,n}(0)$ is also well-defined for all $n \geq 1$ and the integrals are still easily evaluated. However, more care is needed for the case when $b = 1$ since $I_n(1)$ is not well-defined for any n. In this case we redefine the polynomial as an expansion around $\zeta = 1$:

$$p(\zeta) = \sum_{n=0}^{N} d_n (1-\zeta)^n \tag{A.17}$$

Then we have

$$\int_a^b p(\zeta) f_i(\zeta)\, d\zeta = \sum_{n=0}^{N} d_n \big(K_{i,n}(b) - K_{i,n}(a)\big) \tag{A.18}$$

with

$$K_{i,n}(\zeta) = \int (1-\zeta)^n f_i(\zeta)\, d\zeta \tag{A.19}$$

Let $u = 1 - \zeta$. Then

$$K_{1,n}(\zeta) = -\int u^n f_1(1-u)\, du = -\int \frac{u^{n-3/2}}{\sqrt{1-u}}\, du = I_n(1-\zeta) - I_{n-1}(1-\zeta) \tag{A.20}$$

$$K_{2,n}(\zeta) = -\int u^n f_2(1-u)\, du = \int \frac{u^n(1-2u)}{\sqrt{u(1-u)}}\, du = J_{2,n}(1-\zeta) \tag{A.21}$$

$$K_{3,n}(\zeta) = -\int u^n f_3(1-u)\, du = \int \frac{u^n(1-2u)}{\sqrt{u^3(1-u)^3}}\, du = J_{3,n}(1-\zeta) \tag{A.22}$$

(A.23)

Because the mean line is canonical, when $b = 1$, $d_0 = 0$. Therefore the minimum value of n is 1 and $K_{i,n}(1)$ is well-defined.

Finally we must consider the case when we integrate over the full interval [0,1]. In this case the polynomial $p(\zeta)$ must be of the form

$$p(\zeta) = \zeta(1-\zeta)q(\zeta) = \sum_{n=0}^{N-2} a_n \zeta^n \tag{A.24}$$

Let

$$Q_n = \int_0^1 \frac{\zeta^{n-1/2}}{\sqrt{1-x}}\,dx \tag{A.25}$$

It is easy to determine that $Q_0 = \pi$. For $n > 0$ we can integrate by parts to get

$$\begin{aligned} Q_n &= -2\zeta^{n-1/2}\sqrt{1-x}\Big|_0^1 + 2(n-\tfrac{1}{2})\int_0^1 \zeta^{n-3/2}\sqrt{1-x}\,dx \\ &= (2n-1)\int_0^1 \frac{\zeta^{n-3/2}-\zeta^{n-1/2}}{\sqrt{1-x}}\,dx = (2n-1)(Q_{n-1}-Q_n) \end{aligned} \tag{A.26}$$

Therefore

$$Q_n = \frac{(2n-1)Q_{n-1}}{2n} = \frac{(2n)!\pi}{(2^n n!)^2} \tag{A.27}$$

We can now write

$$\alpha_0 = -\frac{1}{\pi}\int_0^1 \zeta(1-\zeta)q(\zeta)f_1(\zeta)\,d\zeta = -\frac{1}{\pi}\sum_{n=0}^{N-2} a_n Q_{n+1} \tag{A.28}$$

$$\begin{aligned} C_m &= 2\int_0^1 \zeta(1-\zeta)q(\zeta)f_2(\zeta)\,d\zeta + \frac{\pi}{2}\alpha_0 \\ &= 2\sum_{n=0}^{N-2} a_n(Q_{n+1} - 3Q_{n+2} + 2Q_{n+3}) + \frac{\pi}{2}\alpha_0 \\ &= \tfrac{1}{2}\sum_{n=0}^{N-2} a_n(3Q_{n+1} - 12Q_{n+2} + 8Q_{n+3}) = -\sum_{n=0}^{N-2}\frac{(n+6)(2n+1)a_n Q_n}{4(n+2)(n+3)} \end{aligned} \tag{A.29}$$

$$\begin{aligned} \alpha_i &= \frac{1}{2\pi}\int_0^1 \zeta(1-\zeta)q(\zeta)f_3(\zeta)\,d\zeta \\ &= \frac{1}{2\pi}\sum_{n=0}^{N-2} a_n(Q_n - 2Q_{n+1}) = -\frac{1}{2\pi}\sum_{n=0}^{N-2}\frac{n a_n Q_n}{n+1} \end{aligned} \tag{A.30}$$

If the mean line is represented by a spline, we can use these expressions to evaluate the integrals on each polynomial segment of the spline. For the case of an arbitrary canonical mean line, we first approximate the mean line with a spline, then use the spline to calculate the integrals.

This page intentionally left blank.

Annex B: Blunt closure of an airfoil

In this annex we describe the algorithm used to close the trailing edge of an airfoil defined using the complete airfoil curve. The curve $\mathbf{x}(\xi)$ is modified so that near the trailing edge it approximates an elliptical arc with radius of curvature r. Let Δ be the trailing edge gap: $\Delta = |\mathbf{x}(1) - \mathbf{x}(0)|$. Then the width of the ellipse is approximately Δ. Let its length be $2a$. Then the radius of curvature at the trailing edge is $r = \Delta^2/4a$ and $a = \Delta^2/4r$.

Let $\hat{t}$ be an aft-pointing unit vector tangent to the mean line at the trailing edge and let $\hat{n}$ be an upward-pointing unit normal to the mean line at the trailing edge. Let X and Y be coordinates aligned with $\hat{n}$ and $\hat{t}$ and with origin at the trailing edge:

$$X = \hat{t} \cdot (\mathbf{x} - \mathbf{x}_{te}); \qquad Y = \hat{n} \cdot (\mathbf{x} - \mathbf{x}_{te}) \tag{B.1}$$

(see Figure B.1). We will also denote by $\big(X(\xi), Y(\xi)\big)$ the value of the ξ-curve in this coordinate system: i.e.

$$X(\xi) = \hat{n} \cdot \big(\mathbf{x}(\xi) - \mathbf{x}_{te}\big); \qquad Y(\xi) = \hat{t} \cdot \big(\mathbf{x}(\xi) - \mathbf{x}_{te}\big) \tag{B.2}$$

Let ξ_p be the value of ξ in the range $[0, \frac{1}{2}]$ such that $X(\xi_p) = -a$. Similarly, ξ_s is the value of ξ in the range $[\frac{1}{2}, 1]$ such that $X(\xi_s) = -a$. It is important to note that if the required radius of curvature is large (so a is small) and the trailing edge gap is oblique to the mean line, one of ξ_p or ξ_s may not be defined: see Figure B.2.

In the region $\xi \in [0, \xi_p]$ we replace the ξ-curve with

$$\mathbf{x}_{new}(\xi) = X_p(\xi)\hat{t} + Y_p(\xi)\hat{n} \tag{B.3}$$

where

$$X_p(\xi) = X(\xi_p) + \big(\eta(\xi) - \xi_p\big)X'(\xi_p) - \big(X(\xi_p) - \xi_p X'(\xi_p)\big)\frac{\big(\eta(\xi) - \xi_p\big)^2}{\xi_p^2} \tag{B.4}$$

$$Y_p(\xi) = -\sqrt{2r\eta(\xi)} + c_p\eta(\xi) + d_p\eta(\xi)^2 \tag{B.5}$$

$$c_p = 3\sqrt{\frac{r}{2\xi_p}} + \frac{2Y(\xi_p)}{\xi_p} - Y'(\xi_p) \tag{B.6}$$

$$d_p = -\sqrt{\frac{r}{2\xi_p^3}} - \frac{Y(\xi_p)}{\xi_p^2} + \frac{Y'(\xi_p)}{\xi_p} \tag{B.7}$$

$$\eta(\xi) = \frac{\xi^2(2\xi_p - \xi)}{\xi_p^2} \tag{B.8}$$

This is continuous and smooth at $\xi = \xi_p$. The function $\eta(\xi)$ is simply a reparametrization which ensures that the derivatives at $\xi = 0$ are well-defined. It is quadratic in ξ

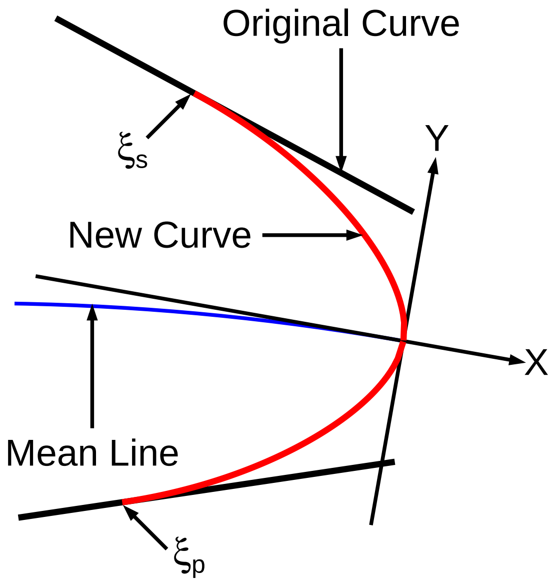

Figure B.1: *Blunt trailing edge closure for an airfoil defined using the complete airfoil curve.*

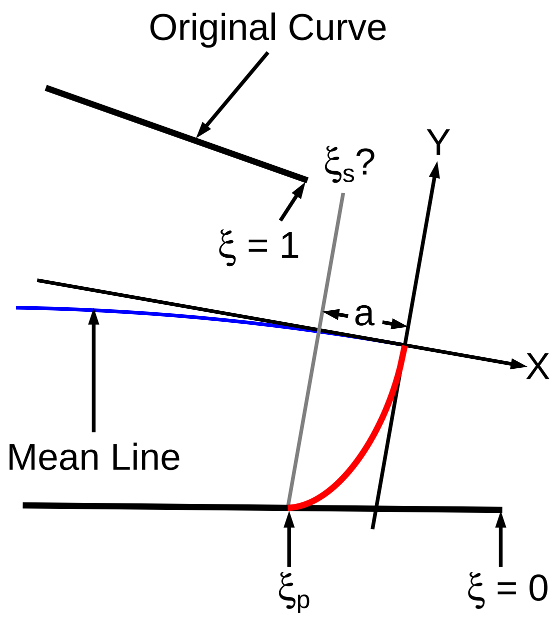

Figure B.2: *A case in which ξ_s is not well defined because the trailing edge gap crosses the mean line obliquely.*

near zero (so that the square root term in Equation (B.5) is linear in ξ) and satisfies $\eta(\xi_p) = \xi_p$ and $\eta'(\xi_p) = 1$ so that it joins smoothly to the standard parameterization at $\xi = \xi_p$.

Similarly, in the region $\xi \in [\xi_s, 1]$ we define

$$\mathbf{x}_{new}(\xi) = X_s(\xi)\hat{t} + Y_s(\xi)\hat{n} \tag{B.9}$$

with

$$X_s(\xi) = X(\xi_s) + \big(\eta(\xi) - \xi_s\big)X'(\xi_s) - \big(X(\xi_s) + (1-\xi_s)X'(\xi_s)\big)\frac{\big(\eta(\xi)-\xi_s\big)^2}{(1-\xi_s)^2} \tag{B.10}$$

$$Y_s(\xi) = \sqrt{2r\big(1-\eta(\xi)\big)} + c_s\big(1-\eta(\xi)\big) + d_s\big(1-\eta(\xi)\big)^2 \tag{B.11}$$

$$c_s = -3\sqrt{\frac{r}{2(1-\xi_s)}} + \frac{2Y(\xi_s)}{(1-\xi_s)} + Y'(\xi_s) \tag{B.12}$$

$$d_s = \sqrt{\frac{r}{2(1-\xi_s)^3}} - \frac{Y(\xi_s)}{(1-\xi_s)^2} - \frac{Y'(\xi_s)}{(1-\xi_s)} \tag{B.13}$$

$$\eta(\xi) = 1 - \frac{(1-\xi)^2(1-2\xi_s+\xi)}{(1-\xi_s)^2} \tag{B.14}$$

Annex C: Prototypes for FormattedOStream

A `FormattedOStream` is an output stream which does some basic formatting of the output characters. A line width can be set so that the output will wrap nicely if a line is too long. An indent level can be set so that all lines are indented on the left.

A `FormattedOStream` is intended for output to a computer terminal.

A `FormattedOStream` adapts an underlying `std::ostream`: i.e. all the output is done via the `std::ostream`; the `FormattedOStream` simply adds spaces and new line characters where appropriate to generate the required formatting.

C.1 Constructors

`FormattedOStream` has two constructors:

`FormattedOStream()`
: The underlying `std::ostream` is `std::cout`. The initial indentation is zero and the number of columns of output is 79.

`FormattedOStream(std::ostream &os)`
: Creates a FormattedOStream which adapts `os`. The initial indentation is zero and the number of columns of output is 79.

C.2 Indentation functions

The following three member functions can be used to control the indentation.

`int get_indent() const`
: Returns the current level of indentation.

`int indent(int i)`
: Sets the level of indentation to `i` and returns the previous value. The new level of indentation will not be negative.

`void rel_indent(int i)`
: Increments the level of indentation by `i`. The new level of indentation will not be negative.

C.3 Line length functions

The following functions can be used to control the line length (number of columns of output).

`int get_num_cols() const`

Returns the current number of columns of output.

`int num_columns(int i)`

Sets the number of columns of output to `i` and returns the previous value. The new value will not be smaller than 1.

`void rel_num_columns(int i)`

Increments the number of columns by `i`. The new value will not be less than 1.

C.4 Inserters

The following insertion operators are defined.

`FormattedOStream& operator<<(const char* s)`

Writes a string formatted using the current indentation and number of columns.

`FormattedOStream& operator<<(const std::string &s)`

Writes a string formatted using the current indentation and number of columns.

```
FormattedOStream& operator<<(
  FormattedOStream& (*f)(FormattedOStream&))
```

An inserter for no-argument manipulators.

`FormattedOStream& operator<<(std::ios& (*f)(std::ios&))`

An inserter for standard no-argument manipulators.

```
template<class T>
FormattedOStream& operator<<(const T &t)
```

Converts `t` to a character string then writes the formatted string. The conversion is done by writing `t` to a `std::ostringstream`; therefore `t` must be writable.

C.5 Other member functions

`unsigned number_of_lines() const`

Returns the number of lines written to the stream.

`std::ostream& base_stream()`
: Returns the stream being adapted.

The following functions are all implemented by applying the corresponding function in namespace `std` to the underlying `std::ostream`:

```
std::ios::fmtflags flags() const
std::ios::fmtflags flags(std::ios::fmtflags f)
std::ios::fmtflags setf(std::ios::fmtflags f)
std::ios::fmtflags setf(std::ios::fmtflags flag, std::ios::fmtflags m)
void unsetf(std::ios::fmtflags mask)
std::streamsize precision() const
std::streamsize precision(std::streamsize p)
std::streamsize width() const
std::streamsize width(std::streamsize w)
char_type fill() const
char_type fill(char_type c)
bool good() const
bool eof() const
bool fail() const
bool bad() const
operator void*() const
bool operator!() const
void flush()
```

C.6 Manipulators

The manipulators `resetiosflags`, `setiosflags`, `setbase`, `setfill`, `setprecision` and `setw` are all defined for a `FormattedOStream`. They are each similar to the corresponding manipulator in namespace `std` applied to the underlying `std::ostream`. In addition, the following special purpose manipulator is defined.

`indent`
: Writes spaces until the indentation level is reached. If the current location is to the right of the indentation, nothing is done.

List of symbols

α	Angle of attack.
α_0	Angle of attack which generates zero lift according to thin airfoil theory.
α_i	The ideal angle of attack.
Δ	The thickness at the trailing edge: $\lvert\mathbf{x}(1) - \mathbf{x}(0)\rvert$.
θ	An angle used to traverse circles in the complex plane defining Joukowski airfoils.
Φ	The complex potential for the flow past a Joukowski airfoil.
ϕ	An angle used to identify the leading and trailing edges of a Joukowski airfoil when represented in the q-plane.
η	Parameter used to avoid zero derivatives at the trailing edge of a Joukowski airfoil.
ξ	Non-dimensional airfoil parameter increasing from 0 on the pressure side, through $\frac{1}{2}$ at the leading edge, to 1 at the trailing edge on the suction side.
ζ	The fractional chord length along the straight line joining the leading edge ($\zeta = 0$) to the trailing edge ($\zeta = 1$).
ζ_j	The value of ζ at which the two segments of a NACA 5-digit mean line offset curve join.
ζ_m	The value of ζ at which a mean line has its maximum deviation from the chord line.
A, B	Constants used when defining Joukowski airfoils.
a	The location of the trailing edge of a Joukowski airfoil in the complex q-plane. Also used for fractional arclength around an airfoil. Also used as a parameter for mean lines with constant load distributions.
c	The cambre of an airfoil: i.e. the maximum deviation of the mean line from the chord line.
C	The centre of the circle defining a Joukowski airfoil in the complex q-plane.
C_L	The lift coefficient.

C_{Li}	The lift coefficient at the ideal angle of attack: the ideal or design lift coefficient.
C_m	The pitching moment coefficient about one-quarter chord.
f_1, f_2, f_3	Functions used to calculate α_0, α_i, C_{Li} and C_m according to thin airfoil theory.
k_1	Parameter used to define NACA 5-digit mean lines.
$\mathbf{m}(\zeta)$	The mean line parameterized using ζ.
$\hat{n}$	A unit normal to the airfoil or mean line.
n_x, n_y	Components of $\hat{n}$.
$\mathbf{p}(\zeta)$	The pressure side of the airfoil surface parameterized using ζ.
p	A complex number used to define Joukowski airfoils. In the p-plane the leading edge is at $(-2a, 0)$ and the trailing edge is at $(2a, 0)$.
q	A complex number used to define Joukowski airfoils. In the q-plane the airfoil surface is a circle centred at C passing through $(a, 0)$.
$\mathbf{s}(\zeta)$	The suction side of the airfoil surface parameterized using ζ.
p	A complex number used to define Joukowski airfoils. In the w-plane the airfoil is canonical.
r	Radius of curvature at the leading or trailing edge.
t	The thickness of an airfoil.
V	The speed of flow at infinity past a Joukowski airfoil.
$v_x - iv_y$	Complex velocity.
$\mathbf{x}(\xi)$	The airfoil surface parameterized using ξ.
$\mathbf{x}_{le}$	The point at the leading edge; equal to $\mathbf{x}\big(\frac{1}{2}\big)$, $\mathbf{p}(0)$ and $\mathbf{s}(0)$.
$\mathbf{x}_{te}$	The point at the trailing edge; equal to $\frac{1}{2}\big(\mathbf{x}(0) + \mathbf{x}(1)\big)$ and $\frac{1}{2}\big(\mathbf{p}(1) + \mathbf{s}(1)\big)$.
z	Complex number used to define Joukowski airfoils. In the z-plane the airfoil surface is the unit circle.

Index

This page intentionally left blank.

Distribution list

DRDC Atlantic TM 2009-053

Internal distribution

1 Author

3 Library (1 paper, 2 CDs)

Total internal copies: 4

External distribution

Department of National Defence

1 DRDKIM

2 DMSS 2

Others

1 Library and Archives Canada
(normally 1 copy) 395 Wellington Street
Ottawa, Ontario
K1A ON4
Attn: Military Archivist, Government Records Branch

1 Director-General
Institute for Marine Dynamics
National Research Council of Canada
P.O. Box 12093, Station A
St. John's, Newfoundland
A1B 3T5

1 Director-General
Institute for Aerospace Research
National Research Council of Canada
Building M-13A
Ottawa, Ontario
K1A OR6

1 Transport Development Centre
Transport Canada
6th Floor
800 Rene Levesque Blvd, West
Montreal, Que.
H3B 1X9
Attn: Marine R&D Coordinator

1 Canadian Coast Guard
Ship Safety Branch
Canada Building, 11th Floor
344 Slater Street
Ottawa, Ontario
K1A 0N7
Att: Chief, Design and Construction

MOUs

6 Canadian Project Officer ABCA-02-01 (C/SCI, DRDC Atlantic – 3 paper copies, 3 PDF files on CDROM)

Total external copies: 14

Total copies: 18

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (The name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.) Defence R&D Canada – Atlantic PO Box 1012, Dartmouth NS B2Y 3Z7, Canada	2. SECURITY CLASSIFICATION (Overall security classification of the document including special warning terms if applicable.) UNCLASSIFIED

3. TITLE (The complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S, C or U) in parentheses after the title.)

C++ classes for representing airfoils

4. AUTHORS (Last name, followed by initials – ranks, titles, etc. not to be used.)

Hally, D.

5. DATE OF PUBLICATION (Month and year of publication of document.) January 2010	6a. NO. OF PAGES (Total containing information. Include Annexes, Appendices, etc.) 80	6b. NO. OF REFS (Total cited in document.) 7

7. DESCRIPTIVE NOTES (The category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered.)

Technical Memorandum

8. SPONSORING ACTIVITY (The name of the department project office or laboratory sponsoring the research and development – include address.)

Defence R&D Canada – Atlantic
PO Box 1012, Dartmouth NS B2Y 3Z7, Canada

9a. PROJECT OR GRANT NO. (If appropriate, the applicable research and development project or grant number under which the document was written. Please specify whether project or grant.) Project 11cd23	9b. CONTRACT NO. (If appropriate, the applicable number under which the document was written.)
10a. ORIGINATOR'S DOCUMENT NUMBER (The official document number by which the document is identified by the originating activity. This number must be unique to this document.) DRDC Atlantic TM 2009-053	10b. OTHER DOCUMENT NO(s). (Any other numbers which may be assigned this document either by the originator or by the sponsor.)

11. DOCUMENT AVAILABILITY (Any limitations on further dissemination of the document, other than those imposed by security classification.)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (Any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution (beyond the audience specified in (11)) is possible, a wider announcement audience may be selected.)

13. ABSTRACT (A brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual.)

A library of C++ classes for representing the geometry of airfoils is described. The classes are based on the CurveLib library for representing differentiable curves. Airfoils that have been represented explicitly include Joukowski airfoils, the NACA 4-digit, 5-digit, 16-series and 6-series airfoils, as well as the DTMB modification of the NACA 66 airfoils commonly used for marine propellers. Generic airfoils can be defined from offsets on the airfoil surface or from offsets defining mean line and thickness curves. Several methods for closing the trailing edges of airfoils with a trailing edge gap are also implemented.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (Technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus identified. If it is not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title.)

Airfoils
NACA airfoils
Joukowski airfoils
Geometry
C++
computer programs